fare elettronica

CULTURA ELETTRONICA APPLICATA

OTTOBRE
n° 244 • Anno 21

LINUX NEI SISTEMI EMBEDDED E REAL-TIME

MIKROBASIC
Generazione di melodie, note, effetti sonori e musica con il PIC

LA TECNICA DDS
Generatore di toni DTMF

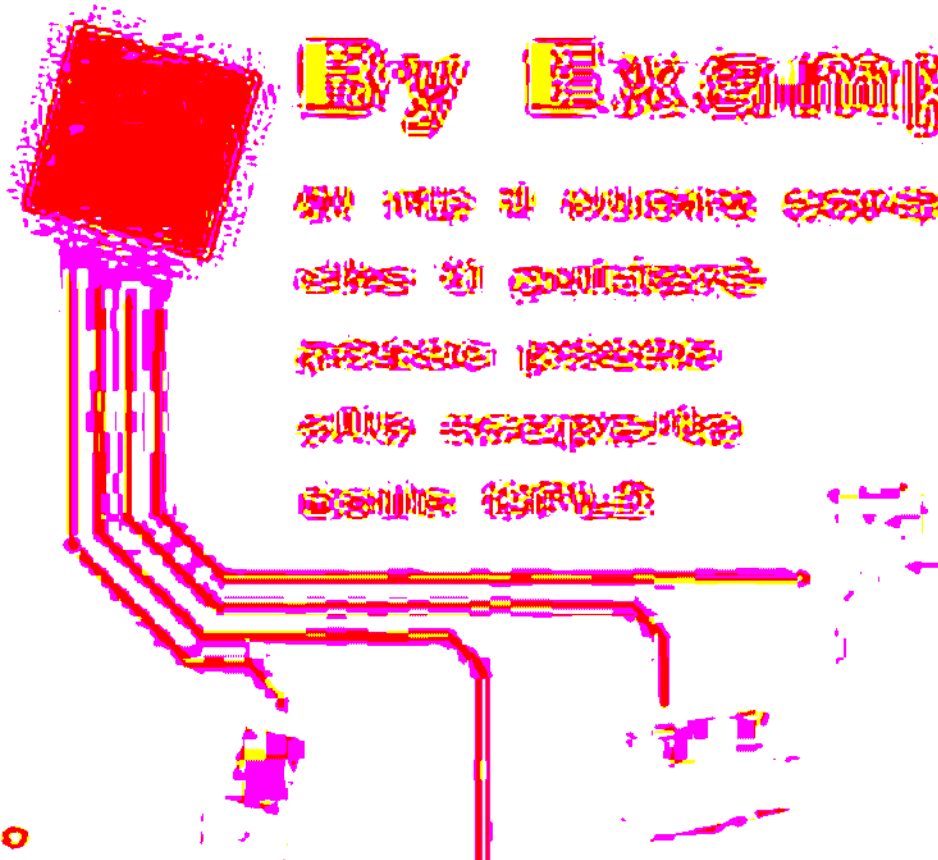
ALIMENTATORI SWITCHING
Switching flyback multi-uscita

VITAMINA C
Tecniche di debug

ELETTRONICANDO

CD-ROM ALLEGATO

Il famoso sistema CAE/CAD in

CPLD By Example

INWARE EDIZIONI

ISSN 1591-2272

€ 5,50

Poste Italiane Spa - Spedizione in abbonamento Postale - D.L. 353/2003 (conv. in L. 27/02/2004 n. 46) art. 1, comma1, DCB Milano. In caso di mancato recapito, restituire all'editore che si impegna a pagare la relativa tassa presso il CPM di Roserio - Milano

Conoscere ed usare...

Quanti gesti facciamo quotidianamente senza renderci conto che fanno parte del nostro bagaglio di conoscenze. Pensate ad esempio alla scrittura, un gesto apparentemente spontaneo, ma che è in realtà il risultato di un percorso iniziato anni prima, dall'apprendimento dell'alfabeto al formulare pensieri coerenti per poi unirli per formare un discorso compiuto. Per tutti, nei primi anni delle scuole elementari, la sfida era semplicemente riuscire a scrivere nello stesso rigo… Il fatto che oggi siamo qui a leggere una rivista di "cultura elettronica applicata" la dice lunga sul percorso fatto da quei quaderni a righe. Leggiamo di microprocessori, di algoritmi, formule e tutto quello che l'elettronica chiede di conoscere per essere plasmata alle proprie necessità. Il fatto che Fare Elettronica si fregi di essere una rivista di cultura elettronica applicata non deriva semplicemente da una decisione presa alla leggera, ma da un percorso durato 20 anni. Un percorso che ci ha portato oggi a proporvi la rivista che tanto apprezzate.
Conoscere ed usare: è questo il trucco. Conoscere a fondo gli strumenti che si hanno a disposizione è l'unica strada per poter raggiungere un risultato certo. L'alternativa è procedere per tentativi… ma quanto costa in termini di tempo e di denaro? E alla fine, a risultato raggiunto, abbiamo imparato qualcosa?
Molti di noi hanno iniziato a fare elettronica semplicemente con un saldatore, lo stagno e un "cercafase", ma quante ore abbiamo speso per capire che il transistor non sarebbe mai andato in conduzione con quella resistenza così alta! L'elettronica è una materia quanto mai vasta, abbraccia tanti campi ed una conoscenza di base è necessaria al fine di raggiungere anche il più piccolo risultato in breve tempo. Ricevo molte lettere di lettori che si congratulano con noi per il fatto di dare tanto spazio alla formazione e che grazie a questo sono riusciti a capire come funziona il circuito che hanno realizzato e sono in grado di modificarlo per adattarlo alle proprie esigenze. Il fatto che abbiamo scelto "Rivista di cultura elettronica applicata" come linea guida di Fare Elettronica e "Conoscere ed usare" come titolo della nostra collana di libri, non è casuale: il nostro scopo è darvi gli strumenti necessari affinché arriviate al risultato con cognizione di causa, non a tentativi.
Ed è proprio per continuare sulla strada che ci contraddistingue, vi proponiamo un nuovo corso: *CPLD By Example*. Le CPLD, ormai largamente utilizzate, rimangono un oggetto pressoché sconosciuto, alla fine di questo corso anche voi potrete utilizzarle con successo nelle vostre applicazioni.

Un numero questo che ha davvero dell'incredibile, ben 16 pagine di contenuti in più rispetto al nostro formato convenzionale ed un bellissimo CD in omaggio.
Infatti, solo Fare Elettronica vi offre in esclusiva la versione *italiana* di **Proteus**, il famoso CAE/CAD professionale. Nel CD trovate due versioni: *LITE* (perfetta per uso hobbistico) e *DEMO*. In questo numero, inoltre, troverete la prima puntata del corso che vi guiderà passo passo all'utilizzo di questo fantastico programma.
Voglio anche segnalarvi la presenza in edicola, a partire dal 15 Novembre, di un bellissimo numero speciale di Fare Elettronica: **100 idee di progetto per 1000 applicazioni**. Questo speciale, **in omaggio per gli abbonati**, è una raccolta di soluzioni circuitali che non può mancare nel vostro laboratorio, prenotate la vostra copia per non rischiare di perderla!

Non mi resta molto spazio per descrivere i restanti articoli che comunque rispecchiano gli alti standard a cui siete ormai abituati, vi auguro quindi una piacevole lettura e vi rinnovo l'appuntamento in edicola a Novembre.

Tiziano Galizia
t.galizia@farelettronica.com

www.farelettronica.com

DIRETTORE RESPONSABILE
Antonio Cirella

DIRETTORE DI REDAZIONE
Tiziano Galizia

DIRETTORE TECNICO
Maurizio Del Corso

HANNO COLLABORATO IN QUESTO NUMERO
Giovanni Di Maria, Riccardo Nicoletti, Salvatore Torrisi, Enrico Raffone, Antonio Di Stefano, Romano Bernarducci, Nico Grilloni, Agostino Rolando, Massimo Divito.

DIREZIONE • REDAZIONE • PUBBLICITÁ
INWARE srl - Via Cadorna, 27/31 - 20032 Cormano (MI)
Tel. 02.66504794 - 02.66504755 - Fax 02.66508225
info@inware.it - www.inwaredizioni.it
Redazione: redazione@farelettronica.com

PROGETTO GRAFICO E IMPAGINAZIONE
Graficonsult - Milano

STAMPA
ROTO 2000 - Via L. da Vinci, 18/20 - 20080, Casarile (MI)

DISTRIBUZIONE
Parrini & C. S.p.a. - Viale Forlanini, 23 - 20134, Milano

UFFICIO ABBONAMENTI
PARRINI & C. S.p.a. - Servizio abbonamenti
Viale Forlanini, 23 - 20134 Milano
Per informazioni, sottoscrizione o rinnovo dell'abbonamento:
abbonamenti@farelettronica.com
Tel. 02.66504794 - Fax. 02.66508225
L'ufficio abbonamenti è disponibile telefonicamente
dal lunedì al venerdì dalle 14,30 alle 17,30

Poste Italiane S.p.a. - Spedizione in abbonamento Postale D.L. 353/2003 (conv. In L. 27/02/2004 n. 46) art. 1, comma1, DCB Milano.
Abbonamento per l'Italia: € 45,00
Abbonamento per l'estero: € 115,00

Gli arretrati potranno essere richiesti, per iscritto, al seguente costo:
Numero singolo: € 7,50
Numero doppio: € 9,00
Numero con allegato: € 8,50

Autorizzazione alla pubblicazione del Tribunale di Milano n. 647 del 17/11/2003.
Iscrizione al R.O.C. n. 11035 19/11/2003

Privacy - Nel caso la rivista sia pervenuta in abbonamento o in omaggio, si rende noto che i dati in nostro possesso sono impiegati nel pieno rispetto del D.Lgs. 196/2003. I dati trasmessi a mezzo cartoline o questionari presenti nella rivista, potranno venire utilizzati per indagini di mercato, proposte commerciali, o l'inoltro di altri prodotti editoriali a scopo di saggio. L'interessato potrà avvalersi dei diritti previsti dalla succitata legge. In conformità a quanto disposto dal Codice di deontologia relativo al Trattamento di dati personali art. 2, comma 2, si comunica che presso la nostra sede di Cormano Via Cadorna 27, esiste una banca dati di uso redazionale. Gli interessati potranno esercitare i diritti previsti dal D.Lgs. 196/2003 contattando il Responsabile del Trattamento Inware Srl (info@inwaredizioni.it).

RICHIESTE DI ASSISTENZA
Per richiedere assistenza o chiarimenti sugli articoli pubblicati, vi preghiamo di contattare direttamente l'autore. Se questo non fosse possibile, utilizzate il modulo di contatto che trovate sul nostro sito web www.farelettronica.com.

COLLABORARE CON FARE ELETTRONICA
Le richieste di collaborazione vanno indirizzate all'attenzione di Tiziano Galizia *(t.galizia@farelettronica.com)* e accompagnate, se possibile, da una breve descrizione delle vostre competenze tecniche e/o editoriali, oltre che da un elenco degli argomenti e/o progetti che desiderate proporre.

ELENCO INSERZIONISTI

Pratica

pag. 68

Teoria

pag. 114

Risorse

pag. 50

Sommario

244
Ottobre
2005

Speciale

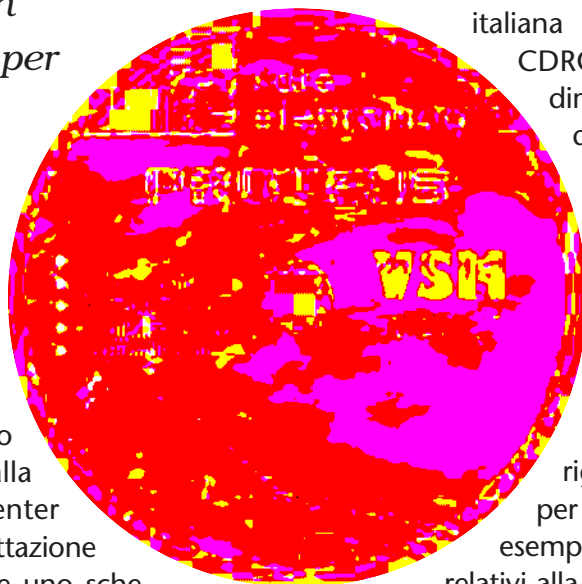
PROTEUS

Lo avevamo annunciato già da diverso tempo e finalmente è arrivato: un nuovo regalo che Fare Elettronica ha pensato per i suoi lettori.
È il CD-ROM di PROTEUS il famoso CAD professionale di progettazione elettronica.
Ecco alcuni suggerimenti utili per l'uso del CD. Più avanti nella rivista troverete un utilissimo tutorial per iniziare subito a lavorare con questo potente strumento.

COSA È PROTEUS

PROTEUS è un pacchetto software, prodotto dalla casa inglese Labcenter Electronics, per la progettazione elettronica comprendente uno schematic capture per il disegno degli schemi elettrici (ISIS), un ambiente per lo sbroglio dei circuiti stampati (ARES) ed una serie di strumenti per la simulazione dei progetti.
ISIS consente il disegno dello schema elettrico in maniera molto semplice grazie alle librerie di oltre 8000 componenti circuitali comprendenti resistori, condensatori, diodi transistors, circuiti integrati e moltissimi componenti interattivi come LED, pulsanti potenziometri e molto altro ancora. Le connessioni elettriche vengono effettuate automaticamente semplicemente cliccando sui due estremi della connessione. Gli strumenti di simulazione vengono utilizzati direttamente in ISIS attraverso numerosi strumenti virtuali quali voltmetri, potenziometri, oscilloscopi ed analizzatori di spettro.
Il passaggio ad ARES è poi di una semplicità estrema: il sistema estrae una netlist che viene automaticamente caricata da ARES. In ARES è poi possibile procedere con lo sbroglio del circuito in maniera automatica o manuale. ARES è dotato di un controllo DRC per la verifica immediata delle regole di progettazione sia fisiche che elettriche.

COSA CONTIENE IL CD

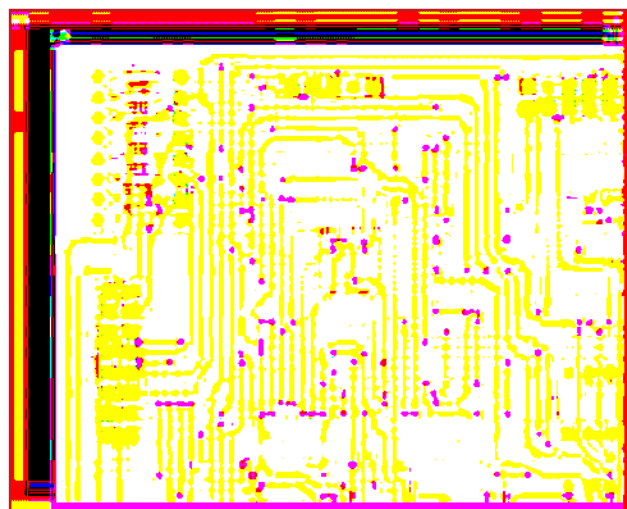
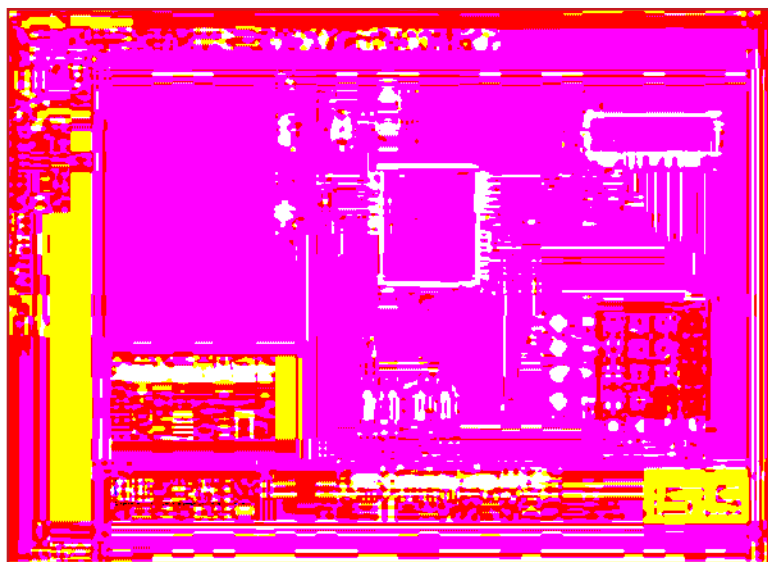
Grazie alla collaborazione tra Fare Elettronica e Labcenter Electronics, è stato possibile proporre ai lettori di Fare Elettronica, l'unica versione italiana del pacchetto PROTEUS. Il CDROM contiene la versione dimostrativa e la versione Lite del programma. La versione demo ha tutte le funzionalità della versione completa con l'unica differenza che non è consentito il salvataggio e la stampa dei progetti. Questa versione consente di valutare tutte le funzioni di PROTEUS sia per quanto riguarda la simulazione, che per lo sbroglio dei circuiti. Tra gli esempi proposti ve ne sono molti relativi alla simulazione con VSM dei circuiti comprendenti microcontrollori. In questo modo è possibile simulare un intero circuito a microcontrollore caricando il programma nel micro stesso.
Nella versione demo è possibile caricare diversi programmi all'interno dei micro, ma non è consentito variare la topologia del circuito.
Per consentire ai lettori di Fare Elettronica di usare questo strumento per la realizzazione dei propri progetti, abbiamo incluso anche la versione Lite. Questa non dispone di tutte le funzioni della versione completa, ma consente il salvataggio e la stampa dei vostri progetti. Trovate la versione Lite nella cartella LITE del CDROM.

il CD-ROM allegato

Ovviamente non potevano mancare i contenuti speciali, per cui troverete le versioni dimostrative dei CD di Fare Elettronica quali le annate 2003 e 2004 e gli altri CDROM preparati per voi.
Non meno importante, la possibilità di acquistare il pacchetto PROTEUS a condizioni vantaggiose.

INSTALLAZIONE DI PROTEUS

Il CDROM è predisposto per l'avvio automatico e qualora non si avviasse, potete lanciare manualmente l'interfaccia grafica cliccando sul file start.exe presente nel CD.
Una volta accettate le condizioni di utilizzo del CD, si accede al menu principale da cui è possibile lanciare l'installazione di Proteus cliccando sulla relativa icona.
Se volete installare la versione LITE, dovrete navigare manualmente il CD e lanciare il file eseguibile presente nella cartella LITE del CDROM.
Essendo la prima versione in italiano, è possibile che vi siano alcune imprecisioni o comunque possibili migliorie. Potete dunque inviare tutti i vostri suggerimenti e le vostre segnalazioni all'indirizzo proteus@farelettronica.com.
Sul sito www.farelettronica.com è stato allestito un forum dedicato a PROTEUS sul quale vi notificheremo gli eventuali aggiornamenti e modifiche relative alla traduzione del programma e che potete usare per scambiarvi idee, trucchi e suggerimenti.

REQUISITI DI SISTEMA

Per utilizzare PROTEUS è necessario disporre di un PC con Windows 2000/XP, 256MB di RAM, 200MB di spazio su disco. Per qualsiasi segnalazione, problema o suggerimento contattate proteus@farelettronica.com.

Scrivete a:
MAILBOX
REDAZIONE DI
FARE ELETTRONICA
Inware s.r.l.
Via Cadorna, 27/31
20032, Cormano (MI)

Oppure inviate un'email a:
mailbox@farelettronica.com

Mailbox

Questa rubrica ospita le richieste più interessanti pervenute dai lettori. Per quanto possibile verrà data risposta a tutte le richieste pervenute via email.

GUITAR FUZZ EFFECT

Sono un appassionato di elettronica e ho anche l'hobby degli strumenti musicali. Vorrei costruire un circuito in grado di generare l'effetto Fuzz per la mia chitarra elettrica. Potreste pubblicare un schema semplice con componenti di facile reperibilità?

Giovanni Calderi

L'effetto Fuzz può essere facilmente ottenuto aggiungendo al segnale di uscita della chitarra elettrica, una serie di non linearità. Nel circuito proposto (figura 1) le non linearità vengono introdotte dai diodi D1 e D2 e dall'operazonale stesso il cui guadagno è così elevato da introdurre distorsioni significative. Il circuito impiega un operazionale LM741 e può essere alimentato con una comune batteria alcalina da 9V. Visto l'esiguo numero di componenti, il circuito può trovare alloggiamento all'interno della chitarra, all'interno dell'amplificatore oppure in un contenitore equipaggiato con uno switch in modo da poter disabilitare l'effetto Fuzz con estrema facilità.

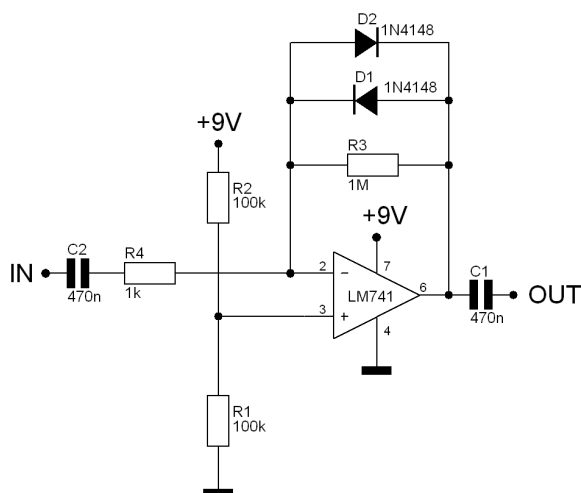



Figura 1 *Fuzz Effect per chitarra*

APPLICAZIONI CON IRF511

Da una vecchia scheda elettronica ho recuperato un buon numero di transistori IRF511. Se possibile vorrei avere alcuni schemi applicativi per questo componente.

Salvatore Canella

IRF511 è un transistore MOSFET di potenza a canale N ad arricchimento in package TO220. In figura 2 è riportata l'identificazione dei terminali ed il relativo simbolo circuitale.
La figura 3 mostra una tipica applicazione come amplificatore audio in classe A. Il trimmer R3 va regolato in modo da leggere metà della tensione di alimentazione tra i terminali di drain e source. Per R2 si consigliano valori compresi tra 22 e 100 Ohm al fine di non superare la massima corrente sopportabile dal FET.
In figura 4 una applicazione come lampeggiatore a due lampade ad incandescenza. Il circuito è un semplice multivibratore astabile in cui le resistenze ed i condensatori determinano

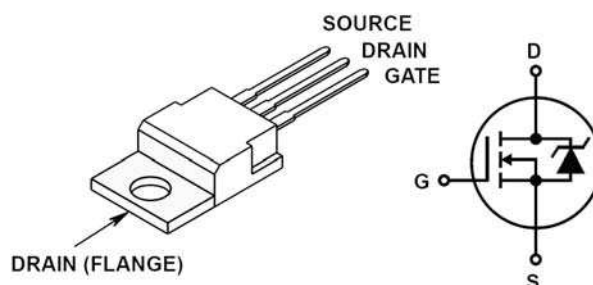



Figura 2 *Il transistore IRF511*

“Richieste, chiarimenti, dubbi e commenti dai lettori”

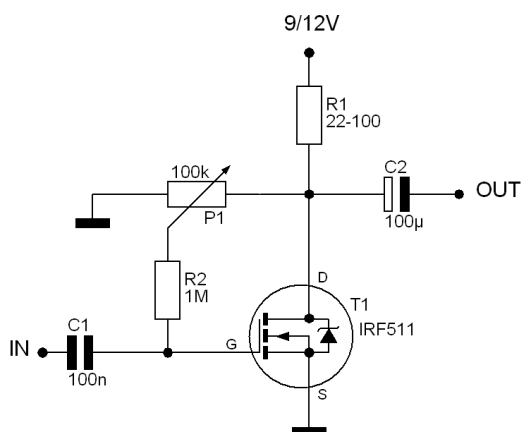



Figura 3 Amplificatore in classe A

la frequenza di lampeggio. Con i valori indicati si hanno circa 3 lampeggi al secondo. Aumentando i valori di R1 e R2 e/o C1 e C2 diminuisce la frequenza del lampeggio.
In figura 5 è mostrata invece una applicazione del IRF511 come interruttore di prossimità. L’elemento sensibile è una placchetta metallica quadrata di circa 7cm di lato.
Mediante il potenziometro P1 si può impostare il livello di sensibilità del circuito quindi determinare l’azionamento del buzzer in funzione della distanza tra il corpo e la placchetta sensibile. Ovviamente il buzzer può essere sostituito con un relé.

Figura 4 Lampeggiatore ad incandescenza

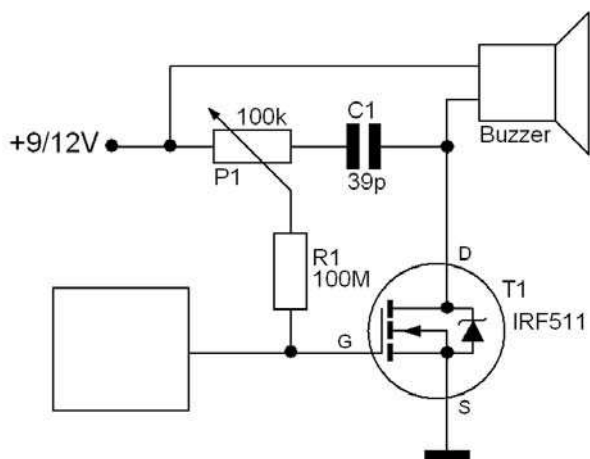



Figura 5 Sensore di prossimità

PROVA TRANSISTOR CON TESTER

Salve, sono un lettore della vostra rivista e ho un problema. Come faccio con il tester a controllare se un transistor di potenza funziona? Forse con il prova diodi ma non so dove mettere i puntali del tester e che cosa dovrei leggere nel tester se il transistor funziona! Siccome sto realizzando un amplificatore preso da un progetto della vostra rivista, con dei transistor usati ho bisogno di capire se sono ancora utilizzabili.

Fabio Cerami

La procedura indicata di seguito è valida per transistor bipolari a giunzione (BJT) siano essi pnp o npn. Ricordiamo che una giunzione pn conduce solo se ai suoi capi è presente una tensione il cui positivo sia sul lato p (anodo) ed il negativo sul lato n (catodo).
Per la prova di un transistor si può quindi utilizzare il tester come provadiodi o come ohmetro. Si devono provare tutte le possibili combinazioni dei puntali sui vari terminali e verificare le situazioni di figura 6 per transistori pnp o npn.
Ricordiamo che la parte a drogaggio diverso è sempre la base (quindi p nel transistor npn ed n nel pnp) per cui se si dispone dell'identificazione dei terminali del transistor risulterà più semplice l'applicazione della procedura.
Nel caso in cui non si conosca l'identificazione dei terminali di base collettore ed emettitore (o si ignori addirittura che il transistor sia un pnp o un npn), si dovrà procedere per tentativi fino a ricondursi ad uno dei due casi indicati. La prova non garantisce comunque l'affidabilità del transistore, ma permette di determinare se una delle due giunzioni è interrotta o è in corto circuito.
Si ricorda che alcuni modelli di multimetro lavorano a polarità invertita (quindi il terminale rosso è il negativo).
Si consiglia di fare una prova su un diodo nuovo (giunzione pn) in modo da stabilire la corretta polarità.

Figura 6 Prova transistor con multimetro

Rubrica di notizie e novità dal mondo dell'elettronica.

Prima pagina

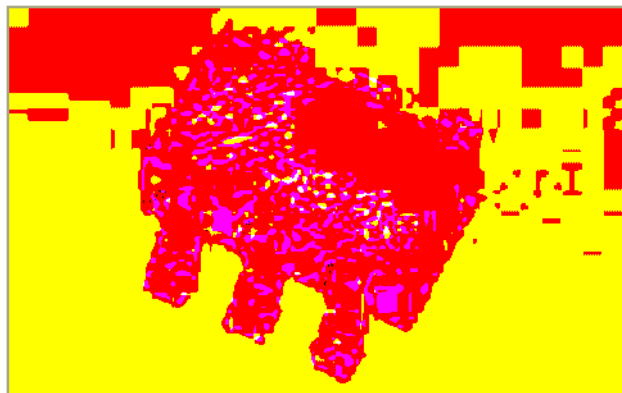
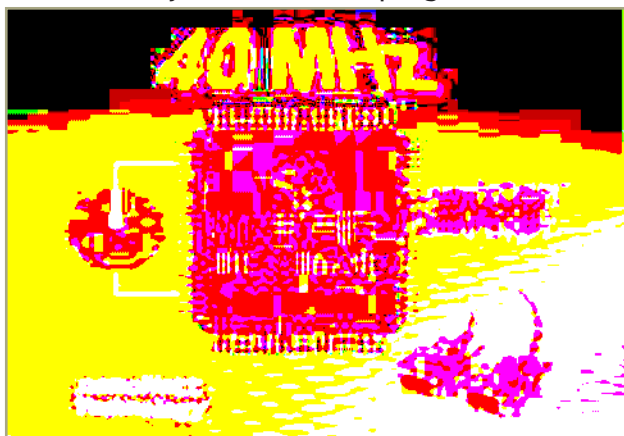
244-01 NUOVA FAMIGLIA DI MICROCONTROLLER PIC® FLASH A 8-BIT

Microchip annuncia i primi dieci modelli della sua famiglia di microcontroller Flash high-pin-count, con memoria ad alta densità PIC18F87J10. I nuovi prodotti raddoppiano le prestazioni low-voltage garantendo fino a 10 MIPS a 3V. Poichè le differenze prestazionali tra microcontroller a 8, 16 e 32 bit continuano a convergere e i progetti tendono a evolvere verso il contenimento dei consumi, gli ingegneri sono costantemente alla ricerca di microcontroller economici a 8-bit ricchi di periferiche e dotati di grandi densità di memoria e di elevati pincount, in grado di aiutarli a preservare gli investimenti fatti in termini di codice a 8-bit e di tool di sviluppo. La famiglia PIC18F87J10 soddisfa queste esigenze offrendo un accesso lineare a fino a 128 Kbyte di memoria Flash onboard e assicurando la compatibilità a livello di codice e di tool con tutti i microcontroller della famiglia PIC18F. Oltre a questo, i nuovi microcontroller PIC18F87J10 includono la tecnologia nanoWatt - la quale garantisce una gestione ottimale dei consumi – nonchè due porte seriali sincrone (per SPI™ o I²C™) e due porte seriali asincrone (USART con supporto LIN).

Tra le altre caratteristiche salienti, spiccano: Da 32 a 128 Kbyte di memorie programma Flash - Fino a 4 Kbyte di RAM dati - Due moduli USART potenziati per l'interfacciamento seriale RS232, RS485 e LIN - Due moduli Master Synchronous Serial Port, ciascuno con supporto per interfacce Master I²C o SPI - Cinque Timer/Counter (3 x 16-bit, 2 x 8-bit) - Analog-to-Digital Converter a 10-bit da 15 canali con temporizzazioni di auto-acquisizione - Due comparatori analogici - Tre moduli potenziati di Capture/Compare/PWM (con supporto drive H-bridge per motori) - Due moduli di Capture/Compare/PWM standard - Watchdog Timer esteso con opzioni prescaler - Reset per Brownout - Accesso a memoria esterna (fino a 2 Mbyte).

244-02 SWITCH RF CMOS CHE OFFRE CONSUMI ULTRA-BASSI

Peregrine Semiconductor Corporation, produttore dei più avanzati RF-CMOS e circuiti integrati mixed signal ad alte prestazioni del settore, ha annunciato oggi il lancio del PE4283, uno switch SPDT riflettente destinato al mercato in crescita della "convergenza" e alle applicazioni RF generiche.

Caratterizzato da bassissimi consumi energetici, punto di compressione elevato e tolleranza ESD eccezionale, il nuovo dispositivo offre una combinazione unica di versatilità, alte prestazioni e prezzo competitivo per applicazioni emergenti quali WLAN, ISM, BT e altri mercati radio a gamma corta. Il PE4283 garantisce meno di 1,0

PER SAPERNE DI PIÙ

Per approfondire le notizie riportate in questa rubrica, visitate il sito **www.farelettronica.com/primapagina** e seguite le istruzioni.
In alternativa potete scrivere a:

Inware Edizioni
Servizio Prima Pagina
Via Cadorna 27 - 20032 Cormano (MI)

Indicando il codice riportato accanto al titolo della notizia (esempio 244-02).

dB di perdita d'inserzione da CC – 4 GHz, migliore di qualsiasi altro switch basato su SOI. Inoltre, la straordinario gestione della potenza e i consumi energetici pressoché nulli per le applicazioni a batteria fanno di questo device il sostituto ideale delle soluzioni pin-diodo in applicazioni che richiedono alle batterie una vita particolarmente lunga e una connettività veramente affidabile. Il PE4283 presenta tolleranza ESD pari a 2kV, il livello più elevato nel settore per uno switch SPDT, e isolamento di 34 dB a 1GHz, un valore eccezionale per uno switch nel sottile package SC-70 a 6 lead. Tra le caratteristiche aggiuntive: punto di compressione P1dB di +32 dBm; bassa perdita d'inserzione di 0,6 dB a 1GHz; logica di controllo CMOS a bordo e ingressi di controllo a pin singolo integrati.

244-03 DISPOSITIVO RGB PLCC-4 LED CON CARATTERISTICHE TERMICHE MIGLIORATE

BivarOpto™, la Optoelectronics Division di Bivar, Inc., ha presentato un nuovo dispositivo a montaggio superficiale (SMT) basato sulla tecnologia RGB per il controllo ottimizzato di colore e luminosità, che include un dispositivo PLCC-4 (plastic leaded chip carrier) standard a basso profilo, offre un bianco brillante e un angolo di

visualizzazione di 120 gradi. Le applicazioni più adatte per questo package molto diffuso sono i display ad alto contrasto quali monitor LCD, retroilluminazione dei cruscotti e gruppi luce esterni degli autoveicoli, retroilluminazione di pannelli strumentali, sistemi di navigazione e commutazione e strumentazione medica.
Il design dell'SMLC RGB presenta un circuito interno a tre chip che utilizza tre die LED a indirizzamento indipendente.
Quando il dispositivo è spento, la lente è trasparente. Il design RGB a tre chip comprende un die AlInGaP e due die InGaN/SIC, con lunghezza d'onda di picco rispettivamente di 625, 568 e 430 nm. Il risultato è un display programmabile bianco o a colori estremamente brillante, in grado di soddisfare esigenze di visualizzazione specifiche. Il valore nominale massimo assoluto della corrente diretta è 200 mA (Peak If). Altri modelli della famiglia SMLC comprendono versioni con chip a uno o due colori: rosso, verde, blu e giallo, e i colori ultra e HE quali il verde puro 395 UV - 525 nm. Altre combinazioni di colore sono disponibili su richiesta.
La configurazione PLCC-4, che comprende una base metallica a 4 tamponi per la dissipazione del calore, ha un'altezza di soli 1,9 mm e un ingombro ridotto: 2,8 mm x 3,5 mm.

244-04 VR STAMP IL NUOVO MODULO PER IL RICONOSCIMENTO VOCALE

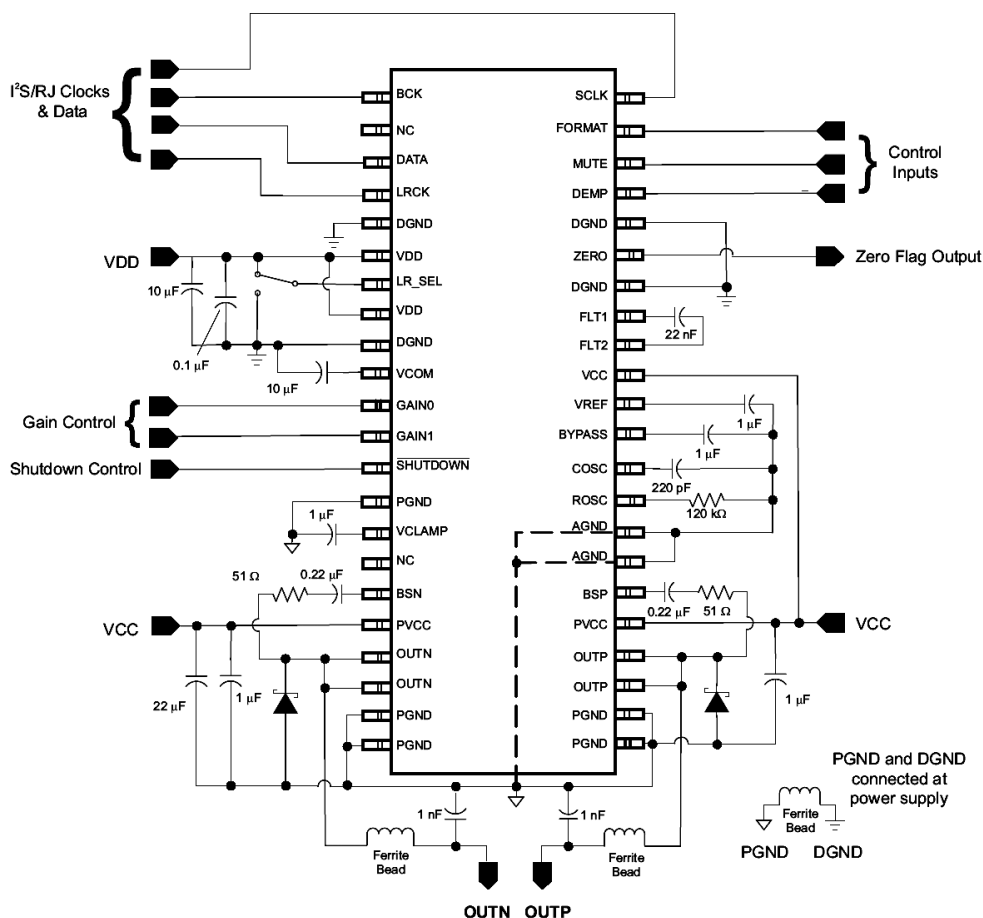
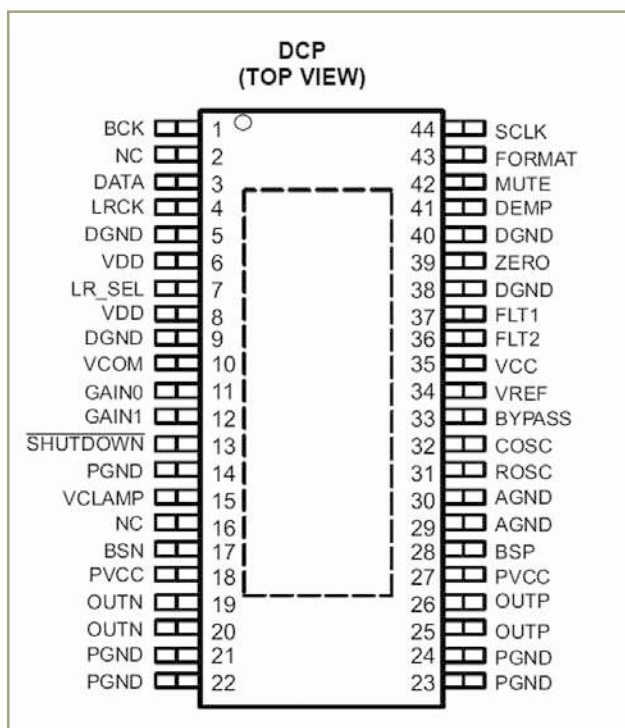
Sensory Inc. ha presentato il nuovo modulo VR Stamp che consente la semplice implementazione di funzionalità per riconoscimento vocale, sintesi vocale e sintesi musicale e touch-tone.
In formato DIP40 VRStamp consente di sfruttare tutta la tecnologia Sensory nel modo più semplice possible.
VR Stamp è basato sul chip RSC4128 di Sensory ed è equipaggiato con una memoria flash da 1Mb.
Offre fino a 24 linee di I/O oltre alle connessioni per il microfono, altoparlante, alimentazione e comunicazione seriale RS232. Il modulo può essere inserito facilmente su qualsiasi scheda prevedendo semplicemente uno zoccolo a 40 pin dual-in-line.
Al momento costituisce la soluzione più semplice ed immediata per aggiungere le funzionalità di riconoscimento vocale nella propria applicazione.
Caratteristiche: tecnologie Speaker Independent e Speaker Dependent con alta immunità al rumore, disponibili i dizioni in diverse lingue tra cui l'italiano, compressione ad alta qualità 2.4-10.8 kbps ed effetti sonori grazie alla tecnologia SX, Speaker Verification (riconoscimento biometrico di password vocali), sintetizzatore musicale MIDI compatibile ad 8 voci e sintesi Touch Tone (DTMF), Audio Wakeup dalla modalità a basso consumo.
Caratteristiche tecniche: RSC-4128 Speech processor & 1Mbit Flash, 128Kb serial EEPROM dati, 14.3MHz (main) & 32KHz (time keeping) clocks, 24 linee I/O, preamplificatore microfonico con AGC, uscita audio in PWM), uscita audio DAC. Consumi: VDD = 2.70V – 3.6V, Iact= 26mA @ 3V, Isleep = <20uA @ 3V.

Dal blocco note di Fare Elettronica una raccolta di idee da tenere sempre a portata di mano.

Notepad

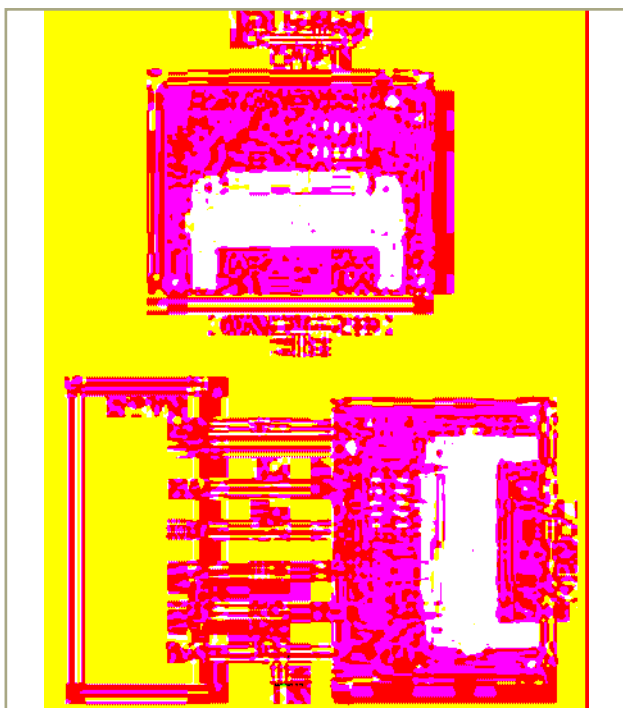
Questa rubrica ha lo scopo di fornire degli schemi applicativi o idee di progetto dei componenti elettronici più interessanti, selezionati per voi dalla redazione. Tutti gli schemi presentati sono elaborazioni di quelli ufficiali proposti dai produttori nella documentazione ufficiale.

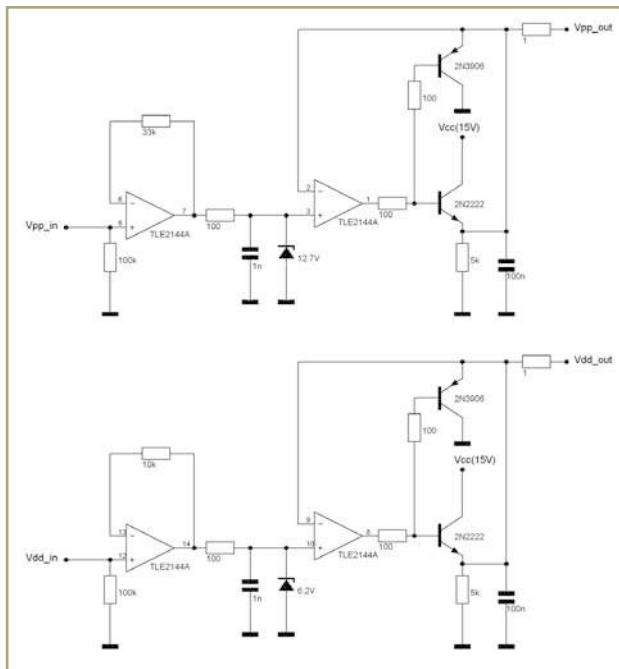
AMPLIFICATORE DA 20W CON INGRESSO AUDIO DIGITALE

Particolarmente adatto come stadio amplificatore per un processore audio, il TPA3200D1 di Texas consente la realizzazione di un amplificatore audio da 20W con ingresso I2S e formato dati a 16Word giustificate a destra. La risoluzione è di 24 bit e la frequenza di campionamento può essere compresa tra 5KHz e 200KHz. La potenza di uscita è di 20W su un carico da 8 Ohm, il guadagno è impostabile fra tre livelli preimpostati ed è prevista una protezione contro corto-circuito in uscita e surriscaldamento eccessivo. In figura il pinout ed uno schema applicativo.

CONTROLLO DI VELOCITÀ PER VENTOLE DI RAFFREDDAMENTO

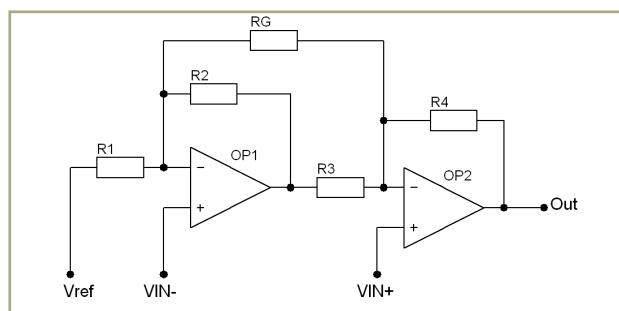
Il TDA21801 è un controllore di velocità per ventole di raffreddamento che necessita di pochissimi componenti esterni per funzionare.
La corrente della ventola viene misurata attraverso un resistore di shunt Rsin.
La temperatura viene misurata attraverso un termistore NTC connesso all'apposito ingresso. In figura la pedinatura ed uno schema applicativo.

CONVERSIONE SERIALE/WI-FI

Con il modulo Anaheim Wi-Fi è possibile operare un conversione da Seriale a Wi-Fi in maniera trasparente ed immediata. Il modulo è dotato di una scheda Wi-Fi in formato Compact Flash che permette la configurazione "Infrastructure network" e "ad hoc network". Nel primo caso è richiesta la presenza di un access point 802.11b mentre nel secondo caso la connessione è punto-punto. Dal lato seriale è possibile lavorare in RS232 o RS485 con o senza transceiver. In figura il modulo ed una sua applicazione con un PIC16F84A.

PROGRAMMAZIONE IN-CIRCUIT DEI PIC16CXXX

Per programmare i PIC16Cxxx sfruttando la programmazione IN-Circuit è necessario garantire che i segnali ICSP abbiano fronti di salita sufficientemente ripidi e che venga fornita corrente sufficiente per la scrittura del dispositivo.
A tale scopo il circuito di figura costituisce un esempio di interfacciamento con il micro soprattutto per quanto riguarda le tensioni di alimentazione Vdd e Vpp. Per i segnali RB6 ed RB7 non è mostrata alcuna particolare circuiteria, ma in alcuni casi potrebbe essere necessario bufferizzare questi segnali. Il circuito riportato deve comunque essere testato nell'intera applicazione dell'utente al fine di determinare se e quanto la circuiteria aggiuntiva influisce sulla velocità ed i livelli dei segnali di programmazione.

AMPLIFICATORE PER STRUMENTAZIONE CON 2 OPAMP

Lo schema circuitale di figura permette di ottenere un amplificatore differenziale per strumentazione utilizzando solamente due amplificatori operazionali. L'elevato valore dell'impedenza di ingresso è garantito dalla configurazione non invertente dei due operazionali, mentre la configurazione invertente per Vref, garantisce un alto valore per la reiezione del modo comune. Il guadagno è dato da:

$$Gain = 1+(R4/R3)+(R4+R1)/RG$$

Lo svantaggio di questa configurazione sta unicamente nel fatto che i due operazionali sono connessi in cascata, per cui i due segnali di ingresso subiscono un ritardo differente e ciò potrebbe causare una distorsione a qualsiasi frequenza anche se nell'intervallo di frequenze in cui operano gli operazionali, tale distorsione è minima.

Elettronicando

Le applicazioni degli amplificatori Operazionali sono davvero innumerevoli, e spaziano dall'elettronica analogica a quella digitale, agli oscillatori, agli strumenti di misura elettronici.
Nel nostro breve corso di elettronica, dopo aver conosciuto un po' più da vicino l'Amplificatore Operazionale descrivendone le caratteristiche fondamentali, presentiamo alcuni semplici circuiti che utilizzano questo componente, aiutandoci con CadLogix per simularne il funzionamento.

CIRCUITI CON AMPLIFICATORE OPERAZIONALE

I circuiti realizzabili con gli amplificatori operazionali sono innumerevoli, ce n'è per tutti i gusti! Dai circuiti amplificatori sommatori ai circuiti integratori e derivatori, ai generatori di forme d'onda, ai generatori di tensione di riferimento, agli stabilizzatori di tensione, ai simulatori di induttanza e moltiplicatori di capacità, circuiti trigger, circuiti comparatori, e chi più ne ha più ne metta.
Apparirà evidente al Lettori che sarebbe impossibile effettuare una rassegna completa nel nostro spazio, ma ci preme presentare alcuni semplici circuiti che, per la loro semplicità ed il loro fascino, colpiscano la vostra curiosità in modo da indurvi a scoprirne altri spinti dalla vostra passione!
Se non espressamente indicato in seguito, ricordate di collegare i pin relativi all'alimentazione dell'A.O., quando disegnate lo schema del circuito per la simulazione con CadLogix.

I PRIMI CIRCUITI FONDAMENTALI

Amplificatore invertente e non-invertente

Nella scorsa puntata abbiamo concluso la panoramica sulle caratteristiche degli AO presentando i due primi circuiti che utilizzano questo componente: *l'amplificatore invertente* (figura 1) e *non-invertente* (figura 2).
Le loro tensioni di uscita sono espresse dalle seguenti relazioni:

$$V_{out} = -\frac{R_2}{R_1} V_{in}$$

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}$$

ed i loro guadagni sono quindi dipendenti dai soli componenti passivi presenti nella rete di

Figura 1 Amplificatore invertente con A.O.

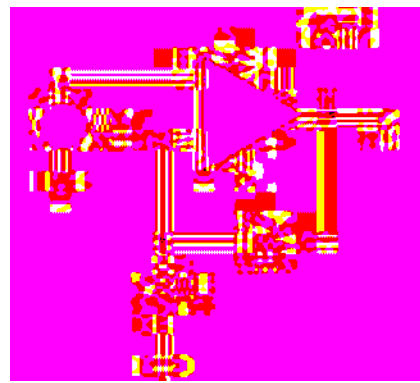
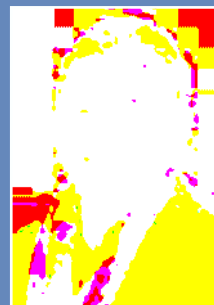



Figura 2 Circuito completo per amplificatore non-invertente con A.O.

Usare gli Amplificatori Operazionali

di Riccardo Nicoletti
(r.nicoletti@farelettronica.com)

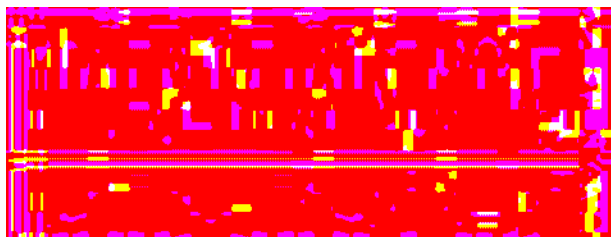



Figura 3 Simulazione dell'amplificatore invertente di figura 1

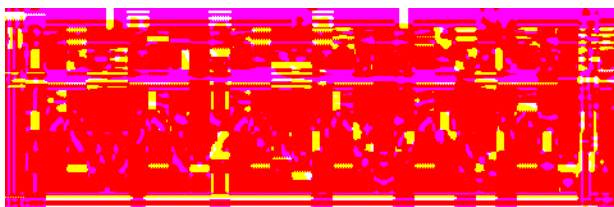



Figura 4 Simulazione dell'amplificatore non-invertente di figura 2

reazione. Abbiamo – ovviamente – evidenziato che il primo circuito sfasa la tensione di uscita rispetto all'ingresso (figura 3), mentre il secondo no (figura 4). Ecco perché il primo si chiama "invertente".

La simulazione è realizzata con R_1=10kΩ ed R_2=100kΩ, quindi Av=-10; in figura 3 è riportata la simulazione per un segnale di ingresso con V_M=0.5V ed f =100Hz.

Per l'amplificatore non-invertente si è scelto R_1=10kΩ ed R2=100kΩ, quindi Av=11; in figura 3 è mostrato il circuito completo per la simulazione, mentre in figura 4 è riportata la simulazione per un segnale di ingresso con V_M=1V ed f=1kHz.

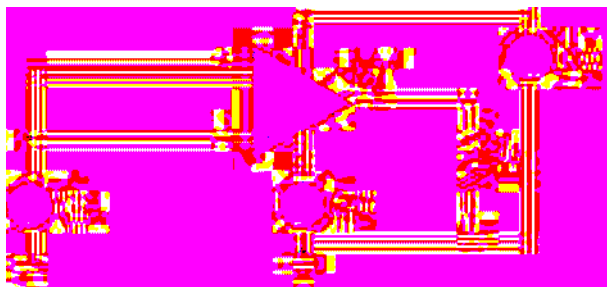



Figura 5 Circuito per la simulazione di funzionamento in modo comune

Figura 6 Simulazione del funzionamento in modo comune (la tensione di uscita è nulla)

ALTRI CIRCUITI CON OPERAZIONALE

Inseguitore

Il circuito inseguitore è disegnato in figura 7. L'uscita "insegue" l'ingresso e quindi si ha sempre V_{out}=V_i, cioè un'amplificazione positiva unitaria. Qual è allora l'utilità di questo circuito? L'amplificatore operazionale possiede una elevata impedenza di ingresso ed una bassa impedenza di uscita; è dunque un *ottimo stadio separatore*, utilizzato come *"buffer"* per separare stadi amplificatori in una catena. Serve comunque in generale quando non si vuole "caricare" l'uscita di uno stadio collegando l'ingresso dello stadio successivo.

Figura 7 Circuito inseguitore con A.O.

In figura 8 è riportato un esempio di funzionamento, ricavato con CadLogix.

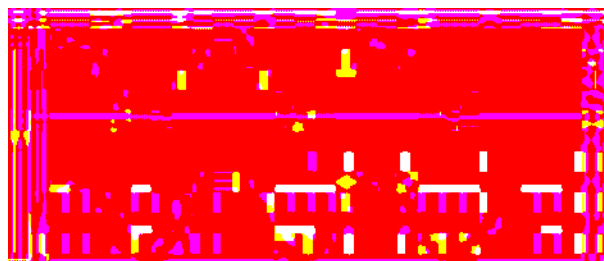



Figura 8 Simulazione di un circuito inseguitore con A.O.

Un parametro importante il CMRR

Il *rapporto di reiezione di modo comune* è un parametro molto importante per un amplificatore operazionale e vorremmo spendere due ulteriori parole per chiarirne il significato.
Tale parametro, introdotto da Nico Grilloni nello speciale FE 238 (p. 44), deve il suo nome alla definizione inglese *"Common Mode Rejection Ratio"* cioè appunto "Rapporto di Reiezione di Modo comune". La dicitura italiana (che potrebbe assomigliare quasi ad un'offesa, se non la si spiega!) fa riferimento al *"modo comune"*. Ma cosa si intende per *"modo comune"*?
L'amplificatore operazionale ha due ingressi, contrassegnati da un simbolo "+" e da "-".
Ai due ingressi possiamo applicare due segnali qualsiasi, tra loro indipendenti. Potremmo anche avere il caso in cui ai due pin viene applicato lo stesso segnale, in modo voluto o non voluto. In questo caso, se i due ingressi vedono lo *stesso segnale*, si dice che l'operazionale funziona in *"modo comune"*, cioè il segnale è *"comune"* ai due ingressi e prende il nome di *"segnale di modo comune"*.
L'operazionale non gradisce lo stesso segnale ai due ingressi, per cui cerca sempre di non amplificarlo e di non farlo sentire in uscita (ecco il significato della parola "reiezione"). Il *guadagno di modo comune* A_{cm} dell'operazionale è molto basso, solitamente può essere di qualche unità.
Questo fatto lo si può simulare con un circuito che possieda ai due ingressi dell'operazionale lo stesso segnale (figura 5).
In figura 6 è riportata la simulazione e si noti il valore della tensione di uscita, praticamente nullo.
L'A.O. gradisce molto segnali diversi applicati sugli ingressi, lavora bene quando la *"differenza"* tra i due segnali è diversa da zero, preferisce quello che si chiama il *"modo differenziale"*.
Il guadagno ad anello aperto A_{OL} è di per sé un "guadagno differenziale". Il CMRR si ottiene facendo il rapporto dei due:

$$CMRR = \frac{A_{OL}}{A_{cm}}$$

e lo si esprime solitamente in decibel

$$CMRR_{dB} = 20 Log \frac{A_{OL}}{A_{cm}}$$

Tanto maggiore è il CMRR di un operazionale e tanto migliore è la sua attitudine ad amplificare segnali differenziali piuttosto che quelli di modo comune.

Rivelatore di picco

Un'applicazione interessante del circuito inseguitore è come parte di un *rivelatore di picco*. Vi ricordate il rivelatore realizzato con diodo e condensatore in FE 239 (Maggio 2005, p. 17)?
L'amplificatore operazionale consente, con le sue caratteristiche, di migliorare le prestazioni del rivelatore; l'inseguitore finale (figura 9) agisce da separatore con lo stadio che segue (nel nostro caso rappresentato da R_L).
Il circuito diodo-condensatore presentato in FE 239 non consente di inseguire segnali di ampiezza inferiore alla $V\gamma$ del diodo, in quanto

Figura 9 Schema di principio per il rivelatore di picco

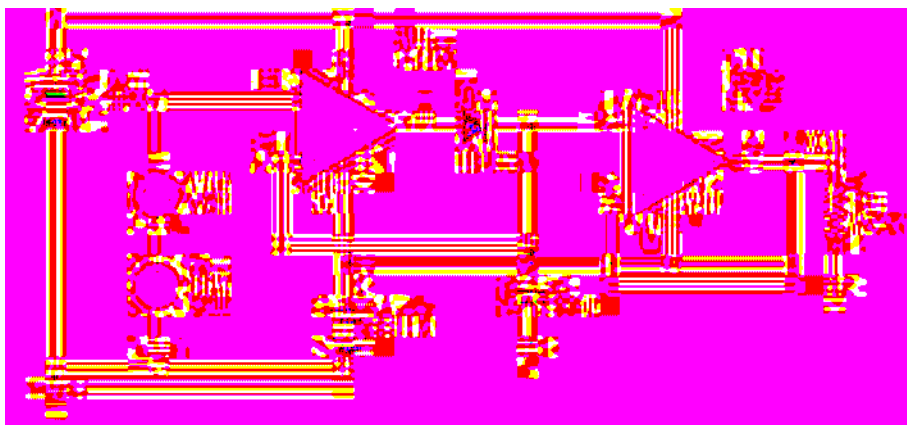



Figura 10 *Circuito rivelatore di picco per simulazione*

un tale segnale non lo porta in conduzione (la caduta ai capi del diodo è inoltre molto influenzata dalla temperatura).
La presenza dell'operazionale annulla la tensione di soglia del diodo, portandolo in conduzione già con $V_{in}>0$. Il diodo collegato all'A.O. come in figura prende il nome di *"diodo di precisione"*.

Il condensatore C viene caricato dalla forma d'onda V_{in} presente in ingresso. Il diodo conduce solo quando la tensione di ingresso V_{in} supera il valore a cui si è caricato il condensatore. Lo stadio inseguitore, con la sua elevata resistenza di ingresso, impedisce al condensatore di scaricarsi e riporta la tensione presente su C direttamente sul carico R_L.
In figura 10 è riportato lo schema completo pronto per la simulazione.

Una cura particolare va prestata nella scelta dei componenti. Senza addentrarci troppo nei dettagli ci preme evidenziare che l'amplificatore operazionale va scelto sufficientemente veloce (con elevato slew-rate) e con basse correnti di bias (un A.O. con ingresso a FET, come l'LF355 o l'OPA111 possono andar bene), il diodo deve avere una corrente di leakage molto bassa ed il condensatore deve possedere basse perdite al fine di mantenere la carica acquisita.
Tale circuito è il nucleo da cui parte la realizzazione di un dispositivo campionatore *Sample&Hold*.
In figura 11 è riportata una simulazione. La forma d'onda inferiore rappresenta la tensione sul carico; il mantenimento del valore può essere ulteriormente migliorato con una opportuna scelta del condensatore.

Figura 11 *Simulazione del rivelatore di picco con A.O.*

Amplificatore differenziale

Il circuito di figura 12 rappresenta un circuito amplificatore di tipo *differenziale*. Utilizzando le "regole d'oro" enunciate per studiare i circuiti

Figura 12 *Circuito amplificatore differenziale*

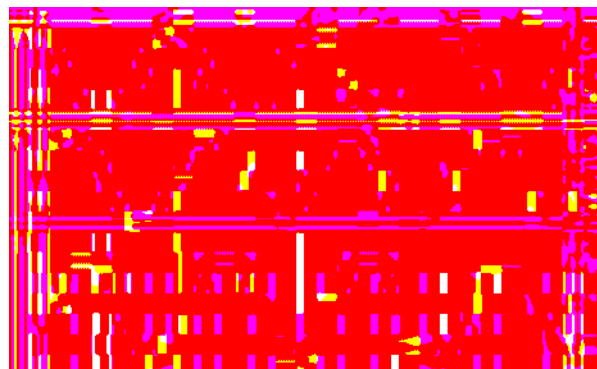



Figura 13 *Simulazione circuito amplificatore differenziale, con V_1 diverso da V_2*

con A.O., ed *applicando il principio di sovrapposizione degli effetti*, potreste provare a ricavare la relazione tra segnale di ingresso e segnale di uscita:

$$V_{out} = \frac{R_2}{R_1}(V_2 - V_1)$$

quindi il circuito amplifica la differenza V_2-V_1 secondo il rapporto R_2/R_1.

In figura 13 si ha una simulazione del funzionamento del circuito di figura 12, quando V_1 è diverso da V_2: la loro differenza è diversa da zero e viene quindi amplificata. In figura 14, invece, è mostrato il caso V_1=V_2 (l'uscita è pressoché nulla, i pochi millivolt che si misurano sono dovuti alla tensione di offset dell'operazionale!).

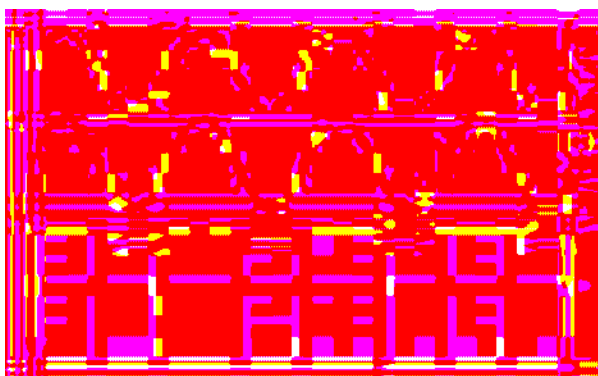



Figura 14 Simulazione circuito amplificatore differenziale, quando V_1=V_2

Amplificatore sommatore invertente

Il circuito di principio è quello di figura 15a.
Analizzando il circuito, applicando le regole d'oro della scorsa puntata, si ricava

$$V_{out} = -\frac{R_S}{R_1}V_1 + \left(-\frac{R_S}{R_2}\right)V_2$$

$$V_{out} = -\frac{R_S}{R_1}V_1 - \frac{R_S}{R_2}V_2$$

e se R_2=R_1

$$V_{out} = -\frac{R_S}{R_1}(V_1 + V_2)$$

quindi il circuito amplifica la somma dei due segnali V_1 e V_2 secondo il rapporto R_s/R_1; il segnale di uscita viene inoltre sfasato di 180°. La figura 15b riporta uno schema pronto per la simulazione; i segnali di ingresso sono due sinusoidi di ampiezza 500mV sfasate tra loro di 90°. Il risultato della simulazione è riportato in figura 16.

Figura 15a Circuito amplificatore sommatore invertente con due ingressi

Figura 15b Circuito sommatore invertente con due ingressi per la simulazione

Il circuito può essere esteso ad N segnali (figura 17), in tal caso se R_1=R_2=R_3=....=R_N si ha

$$V_{out} = -\frac{R_S}{R_1}(V_1 + V_2 + V_3 + \ldots + V_N)$$

se invece le resistenze R_1, R_2, ecc. sono diverse la tensione di uscita è una funzione pesata dei segnali di ingresso.

$$V_{out} = -\frac{R_S}{R_1}V_1 - \frac{R_S}{R_2}V_2 - \frac{R_S}{R_3}V_3 - \frac{R_S}{R_4}V_4 + \ldots - \frac{R_S}{R_N}V_N$$

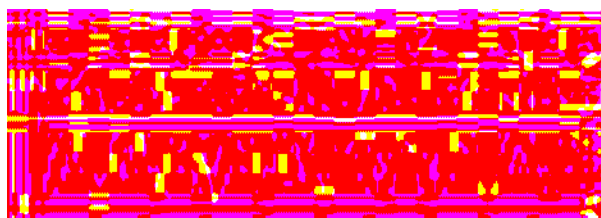



Figura 16 Simulazione per un circuito amplificatore sommatore invertente

Un semplice convertitore digitale-analogico

Un caso interessante del circuito di figura 17 lo si ha quando gli ingressi sono quattro (figura 18). Supponiamo che R_S=10 kΩ e che le quattro resistenze di ingresso siano R_1=1.25 kΩ, R_2=2.5 kΩ, R_3=5 kΩ ed R_4=10 kΩ. Con questi valori si ha (figura 19):

$$V_{out} = -(V_4 + 2 \cdot V_3 + 4 \cdot V_2 + 8 \cdot V_1)$$

I quattro segnali di ingresso potrebbero essere i quattro bit di un numero binario, con V_4 bit meno significativo e V_1 bit più significativo.

Consideriamo il numero 12, in binario 1100; poniamo V_1=1 V, V_2=1 V, V_3=0 V, V_4=0 V. Otteniamo:

$$Vout = -(0 + 2 \cdot 0 + 4 \cdot 1 + 8 \cdot 1) = -12\ V$$

quindi, a parte il segno, la tensione di uscita è il corrispondente analogico della cifra posta in ingresso (figura 20). Questo circuito è un primo esempio di *convertitore digitale-analogico*.

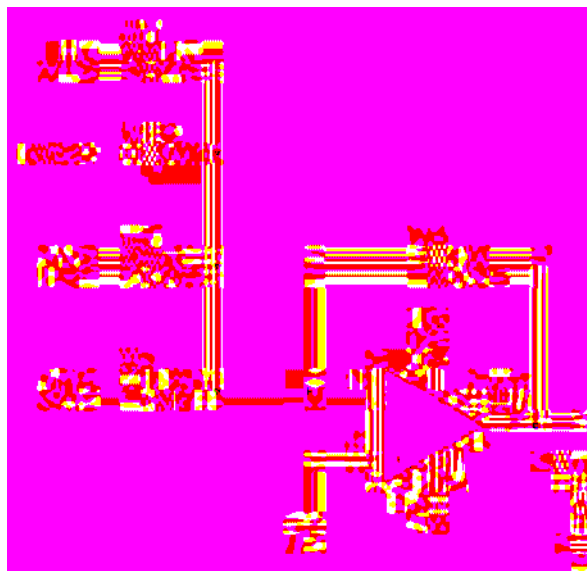



Figura 17 *Circuito amplificatore sommatore invertente con N ingressi*

Convertitore tensione-corrente

Nello schema di figura 21 è riportato un circuito che converte un generatore reale di tensione in un generatore ideale di corrente. Il generatore reale di tensione è rappresentato dal gruppo V-R collegato al morsetto “+” dell’operazionale.

Per le caratteristiche dell’A.O. la tensione V erogata dal generatore si ritroverà sul morsetto invertente per cui farà scorrere in R_1 la corrente (come in figura 21) che attraverserà anche R_2, indipendentemente dal valore di quest’ultima, data l’infinita resistenza di ingresso dell’operazionale.Il valore della corrente dipende solo da R_1 poiché:

$$I = \frac{V}{R_1}$$

e quindi la corrente I non risente della resistenza connessa in serie col generatore V.

Figura 18 Circuito amplificatore sommatore invertente con quattro ingressi

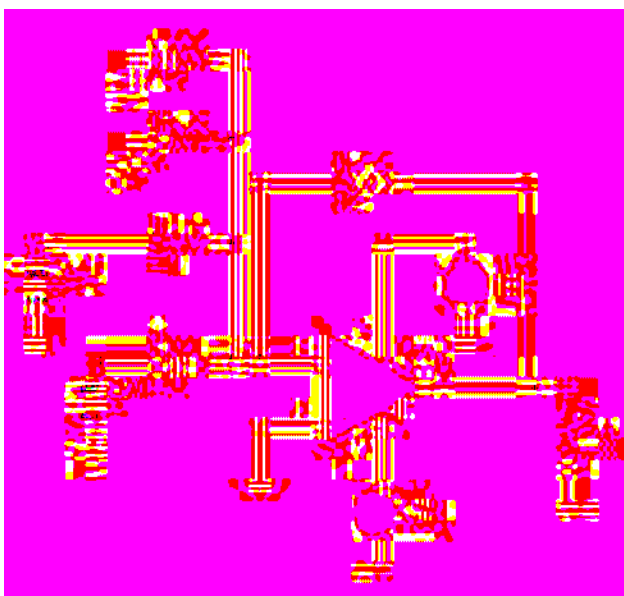



Figura 19 Un primo convertitore digitale-analogico con A.O.

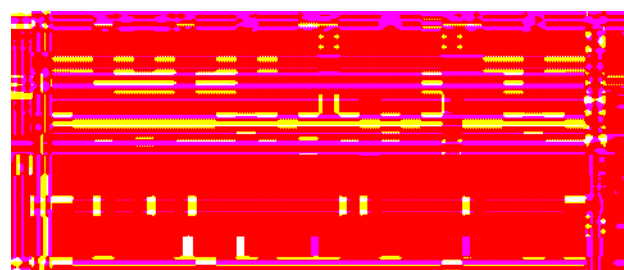



Figura 20 Simulazione dell’esempio con i quattro bit “1100” che in uscita forniscono -12V

Circuito integratore invertente

Il circuito integratore fornisce in uscita una tensione V_{out} che rappresenta, nel tempo, l’integrale del segnale di ingresso V_{in}.

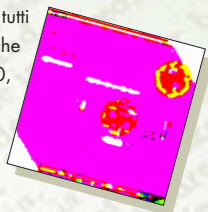
Figura 21 Convertitore da generatore reale di tensione a generatore ideale di corrente

Figura 22 Amplificatore integratore invertente

CD-Rom e Data Book ECA

La migliore Documentazione Tecnica

ECA-403 VRT-DISK 2005

Contiene 120.000 dispositivi differenti (transistor, diodi, tiristori e IC), includendo i dati salienti, il costruttore e quando possibile l'appropriato equivalente. In aggiunta alla descrizione dei pin di tutti i semiconduttori discreti, questo cdrom contiene anche la descrizione dei pin per tutti i CMOS 4000/7400, TTL 7400, molti amplificatori operazionali e alcuni IC audio e video.

ECA-407 LIN-DISK 2003

Contiene il database e la tavola degli equivalenti degli amplificatori operazionali, comparatori, stabilizzatori e regolatori. Un potente motore di ricerca permette di interrogare il database cercando non solo per tipo ma per costruttore, per contenitore, pinout e voltaggio.

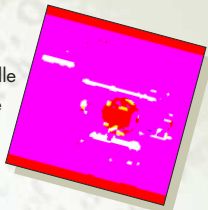
ECA-404 MEM-DISK 2000

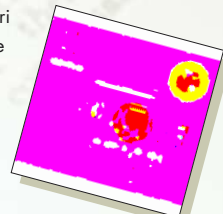
Più di 50.000 memorie differenti, come dRAM, sRAM, EPROM, EEPROM, FIFO e vRAM, con tutti i necessari valori e caratteristiche, pin-outs, tavole della verità, disegno del contenitore e costruttore (con indirizzo). Il programma consente la creazione di 5 data-base personali, dove memorizzare i risultati delle ricerche, questo è utile per la comparazione dei componenti.

ECA-408 OPTO-DISK 2003

Contiene il database e la tabella degli equivalenti per i trasmettitori (LED, IRED, Laser diodi, barre LED e array), ricevitori (fotodiodi e transistor, foto resistenze, foto ICs, foto elementi, celle solari e pyrodetectors), fotoaccoppiatori (fotodiodi, transistor e darlington, digital Ics, amplificatori, foto FET, SCR e foto resistenze), foto relay, foto interruttori, sensori a riflessione (foto sensori) e coppie di trasmettitori e ricevitori.

ECA-405 CMOS/TTL-DISK 2003

Più di 85.000 circuiti integrati digitali, per i quali sono specificati caratteristiche e valori massimi. Il semplice click del mouse, sul componente selezionato, farà apparire: lo schema interno, le dimensioni del contenitore, specifiche dettagliate delle funzioni, tavola della verità, informazioni sulle applicazioni, i contenitori disponibili, il costruttore (completo di indirizzo) e il simbolo secondo lo standard IEEE 91.

ECA-409 DDV-DISK 2002

Oltre 47.000 diodi e tiristori con le loro valori massimi consentiti, pin-out, dati del costruttore e dimensioni.

ECA-406 TDV-DISK 2003

Oltre 100.000 transistor e FET, da A...Z, 2N21...2N7228 fino a 2...40 000...μ. Sono indicati i valori massimi e le caratteristiche, l'assegnazione dei pin, il costruttore e le dimensioni. Oltre alla ricerca per "tipo", il programma consente una "ricerca selettiva" in accordo con alcuni dati importanti, quali "potenza" e "voltaggio".

DATA BOOK ECA 2005

VRT book è la versione cartacea in due volumi del VRT-Disk. Contiene le tabelle comparative di oltre 130.000 componenti tra Transistori, Tiristori, Diodi, circuiti integrati, ecc... **Audio AMP** è un nuovo data-book sugli amplificatori audio con oltre 3500 circuiti diversi.

Figura 23 Circuito completo per la simulazione dell'integratore invertente

Il circuito è utilizzato nel generatori di forma d'onda e negli strumenti elettronici di misura, in quanto consente di ottenere una rampa da un segnale costante, una parabola da una rampa, e così via. Tralasciamo approfondimenti su questa operazione.
Un circuito di principio è riportato in figura 22.

$$V_{out} = -\frac{1}{RC} \cdot \int_0^t Vi(\tau)d\tau$$

In figura 23 è mostrato lo schema completo, pronto per la simulazione. Se in ingresso viene posto un segnale costante (es. V_{in}=2 V), l'uscita avrà la forma di una rampa (figura 24) con pendenza negativa (a causa del segno di V_{out}).
Se la tensione di ingresso è un'onda quadra, in uscita si ricava un'onda triangolare (figura 25).

Circuito derivatore invertente

Il circuito derivatore compie l'operazione inversa rispetto all'integratore. L'operazione di *"derivazione"* è l'inversa di quella di *"integrazione"*. Se vi ricordate, ci siamo imbattuti in questa operazione in FE 236 (Febbraio 2005, p. 26) quando parlavamo dei condensatori.
Se integrando una funzione costante si ottiene una rampa, derivando una rampa si ottiene di nuovo una funzione costante; integrando un'onda quadra si ottiene un'onda triangolare, mentre derivando un'onda triangolare si ricava un'onda quadra. Provare per credere!
Il circuito derivatore (figura 26) fornisce una tensione di uscita che verifica la relazione:

$$V_{out} = -RC \cdot \frac{dV_{in}(t)}{dt}$$

Uno schema di circuito derivatore è quello di figura 26. In figura 27 è mostrata una simulazione. L'ingresso V*in* è una forma d'onda quadra; l'operazione di derivazione, applicata ad un'onda quadra, consente di rivelarne i fronti di salita e di discesa, fornendo un impulso per ogni fronte (l'impulso è negativo per i fronti di salita e positivo per quelli di discesa). Se si vuole intuire l'utilizzo di un circuito simile, pensiamo che questi impulsi in uscita ci consentono di "contare" i fronti di onda quadra: se eliminassimo (con un diodo) i picchi negativi potremmo contare i periodi di onda quadra trascorsi.

I circuiti derivatori sono molto utilizzati come dispositivi *"marcatori"* per individuare fronti d'onda ripidi, ad esempio nelle basi dei tempi degli oscilloscopi e negli strumenti elettronici di misura.

APPLICAZIONE COME COMPARATORE

In assenza di reazione, un A.O. può essere utiliz-

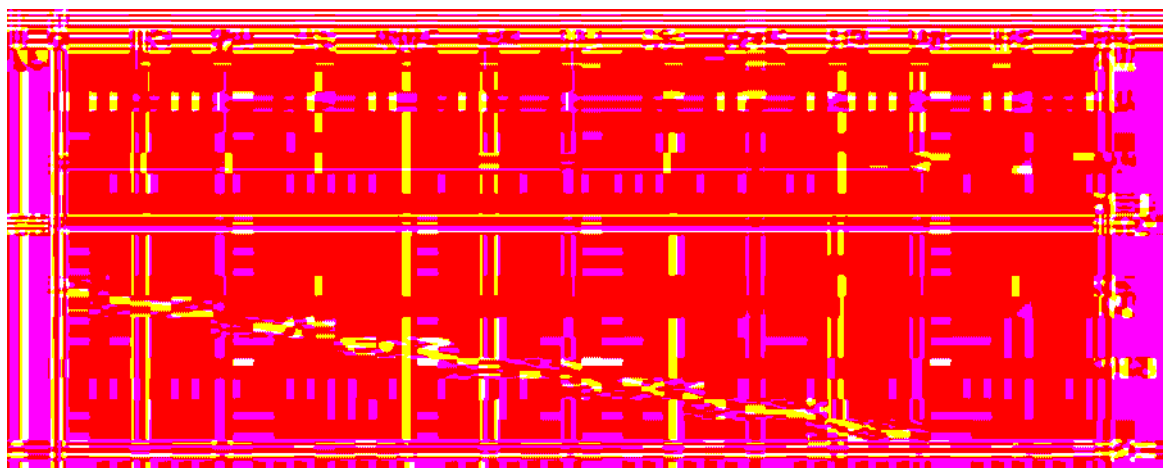



Figura 24 Simulazione dell'integratore invertente, con tensione di ingresso costante

zato come comparatore, in grado cioè di rivelare la minima differenza esistente tra i due ingressi e di distinguere quale dei due ingressi si trova a potenziale maggiore. Esistono dispositivi appositi per tale funzionamento, come ad esempio l'LM339 della National Semiconductor, denominato "Low Power Low Offset Voltage Quad Comparators" e contenuto quindi in un chip in quattro esemplari. Il funzionamento del comparatore è molto semplice: se $V+>V-$ allora $V_{out}=+V_{cc}$, viceversa se $V+<V-$ allora $V_{out}=-V_{cc}$. Vi consiglio di dare un'occhiata al datasheet che trovate nel sito della Rivista, poiché contiene gli schemi di molte applicazioni tipiche davvero interessanti. Un esempio di applicazione con comparatore è il circuito in figura 28a, dove V_{out} è presa ai capi di R_L.

Il pin 7 (ingresso "+") del dispositivo è mantenuto ad un potenziale fisso $V_{rif}=9V$ tramite una

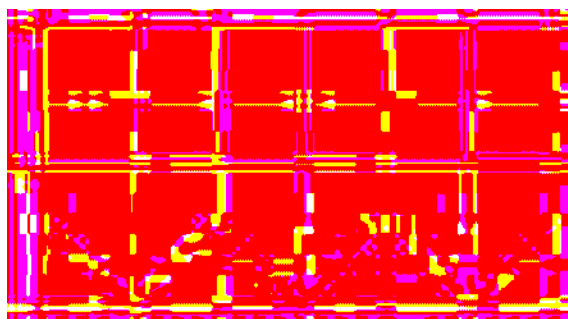



Figura 25 Simulazione dell'integratore invertente, con ingresso ad onda quadra

Figura 26 Amplificatore derivatore invertente

batteria. Il pin 6 (ingresso "-") è collegato ad un condensatore inizialmente scarico; il potenziale dell'ingresso "-" coincide quindi con la tensione V_c. Inizialmente, all'accensione, essendo il condensatore scarico, si ha V+>V- e quindi $V_{out}=+V_{cc}=15V$. Poi il condensatore comincia a caricarsi tramite la resistenza R=1kΩ; la rete RC ha costante di tempo τ=RC=1 s e il potenziale V_c raggiunge il valore V_{rif} dopo circa 920 ms; in tale istante V- supera V+ ed in uscita si ha la commutazione $V_{out}= -V_{cc}$.
Nel circuito di figura 28b, provate a sostituire la rete RC collegata al morsetto "-" con un generatore di forma d'onda sinusoidale di ampiezza 12 V.

Figura 27 *Simulazione amplificatore derivatore invertente*

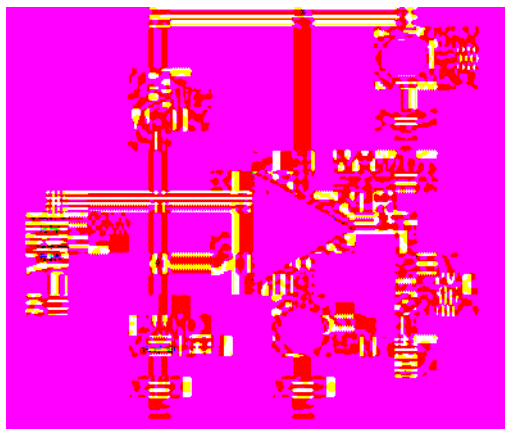



Figura 28a *Circuito comparatore con A.O.*

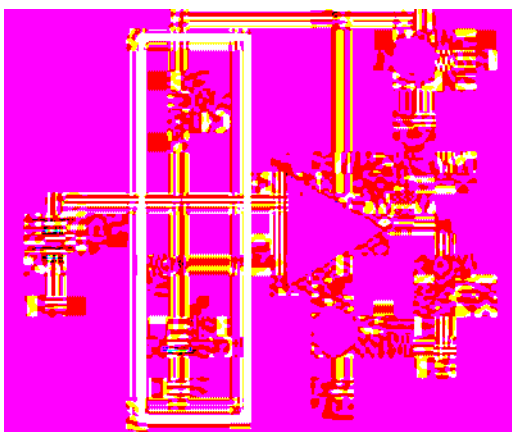



Figura 28b *Rete RC nel circuito comparatore con A.O.*

Speriamo che questa puntata, che non esaurisce il panorama delle applicazioni degli A.O. (non si prefiggeva infatti tale obiettivo!), sia servita a stimolare la vostra curiosità sulle immense ed affascinanti caratteristiche di questo incredibile dispositivo.

PROSSIMA PUNTATA

Nella prossima puntata ci occuperemo di un componente molto famoso: il 555. Tale dispositivo, dalle applicazioni svariate, sarà illustrato nelle sue configurazioni principali. Come al solito il simulatore CadLogix, del quale apprezziamo in ogni puntata la potenza e la semplicità d'uso, ci aiuterà nella comprensione del suo funzionamento.

BIBLIOGRAFIA

Molte applicazioni degli amplificatori operazionali sono illustrate magistralmente in:

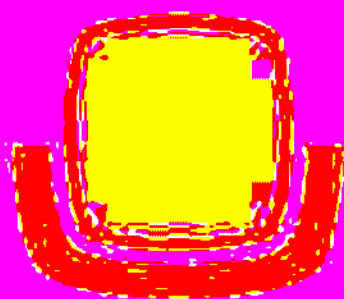
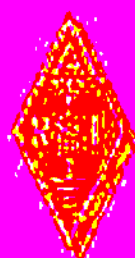
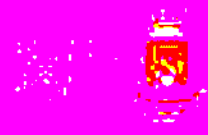
P. Horowitz, W. Hill – *"The Art of Electronics"*, Cambridge University Press, II ed., 1989.

Una buona trattazione è anche contenuta in:

J. Millman, A. Grabel – *"Microelectronics"*, McGraw-Hill International Ed., II Ed., 1987.

Un riferimento irrinunciabile sono in questo caso i manuali dei costruttori; segnalo in particolare:

Texas Instruments – *"Amplifiers and Comparators Data Book"*, 2000 che contiene numerose appendici riguardanti le definizioni dei parametri elettrici, nonché numerosi circuiti applicativi commentati.

Figura 29 *Simulazione circuito comparatore*

La tecnica DDS

Un generatore di toni DTMF standard con selezione a tastiera è presentato in quest'articolo come applicazione della tecnica DDS (Direct Digital Synthesys), mediante un microprocessore. Caratteristiche principali del progetto sono l'utilizzo di un microprocessore della famiglia 16FXX, il numero esiguo di componenti esterni e di linee di programma.

La tecnica DDS, della quale abbiamo ampiamente parlato nel numero 240 (Giugno 2005) di Fare Elettronica, nasce come soluzione "digitale" ai problemi tipicamente di natura analogica quali la generazione d'alte frequenze con elevata stabilità e con altissime risoluzioni che hanno evidenziato i limiti dei circuiti PLL, VCO, ecc. In questo articolo è presentato il primo progetto che utilizza questa tecnica, per mezzo di un PICmicro, nell'ambito della bassa frequenza: ***generatore di toni DTMF con selezione a tastiera.***

IL GENERATORE DI TONI DTMF CON TECNICA DDS

Le caratteristiche principali del circuito, presentato in seguito, sono i pochi componenti esterni necessari al funzionamento del micro 16F84A e il firmware piccolo in termini di righe programma. Il cuore del progetto è la routine DTMF_DDS in cui s'implementa la tecnica DDS per la generazione dei toni DTMF. Altre routine sono usate nel progetto come la routine

F1\F2	1209	1336	1477
697	1	2	3
770	4	5	6
852	7	8	9
941	*	0	#

Tabella 1 *Matrice delle frequenze (in Hz) per toni DTMF standard*

TAST.asm per la gestione della tastiera 4x3
I toni generati sono quelli standard DTMF. Le frequenze (in Hz) relative alle righe e alle colonne sono mostrate nella tabella 1.

Caratteristiche tecniche del circuito (XTAL = 8 MHz)

Tensione d'alimentazione: 2.7-5V
Frequenza Toni: DTMF standard
Durata toni: Variabile
Precisione della generazione: ±1.7 Hz
Altre caratteristiche elettriche sono riportate sul datasheet del PIC16F84A.

Descrizione del progetto

In figura 1 è riportato lo schema elettrico del generatore di toni DTMF a tecnica DDS. Il circuito è costituito dal PIC16F84A e da pochi componenti passivi. Per la sua realizzazione possono essere usati altri microcontrollori della famiglia PIC16FXX.

Funzionamento del circuito

Quando un tasto della tastiera T1 è premuto, la routine di gestione *TAST.asm* ricava il numero, la riga e la colonna relative al tasto premuto. Queste informazioni sono gli ingressi della routine *DTMF_DDS.asm* che genera in uscita un tono multifrequenza corrispondente. La durata del tono in uscita è settabile tramite delle costanti da fissare all'interno della routine. In dettaglio, il microcontrollore implementa due

Generatore di toni DTMF

di Salvatore Torrisi
microst@microst.it

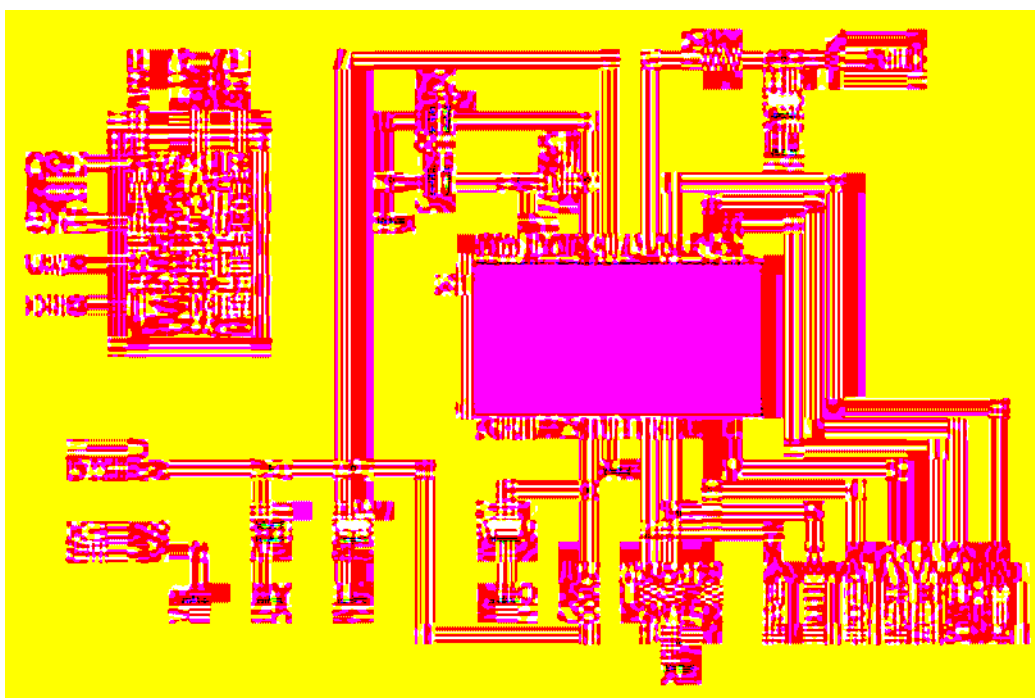



Figura 1 *Schema elettrico del generatore di toni DTMF*

accumulatori di fase a 16 bit. I bit 15 dei singoli accumulatori sono convogliati mediante una funzione "OR" nella stessa uscita (pin RB7 della porta B) che rappresenta l'uscita del circuito. In cascata è applicato un filtro *LPF* che elimina le componenti in frequenza non desiderate.

La tastiera

La tastiera utilizzata per il circuito è una semplice tastiera a matrice con tre colonne e quattro righe. Si possono utilizzare altre tastiere, anche a 5 righe e 4 colonne, avendo cura di mantenere le corrispondenze tra tasti, riga e colonna.

Montaggio

Il montaggio del circuito è estremamente semplice e nessun particolare accorgimento deve essere preso.
Le figure 2 e 3 riportano il circuito stampato lato rame e lato componenti del generatore. Il circuito può essere parte integrante di progetti più complessi quindi il circuito stampato è utilizzabile per la realizzazione del solo generatore.

Dopo aver realizzato il circuito stampato e saldato i vari componenti (si consiglia di utilizzare uno zoccolo per il PIC) si passa alla programmazione del micro mediante i ben noti software e programmatori.

Taratura e Collaudo

La taratura del circuito è semplice ma importante per la qualità della generazione. Per la taratura è necessario avere un frequenzimetro. Dopo aver collegato il probe del frequenzimetro sul pin 15 del PIC, mediante un cacciavite per RF, bisogna agire su C4 ruotandolo in modo da leggere la frequenza del quarzo utilizzato più vicina al suo valore nominale. Se per qualche motivo non si riesca ad ottenere il valore del quarzo, si può cambiare i valori di C3 o C4.
Per ascoltare i toni in uscita si può collegare

l'uscita ad un piccolo amplificatore (le casse del PC vanno bene).
Per testare la qualità dei toni generati collegare l'uscita ad un circuito decodificatore. A tal scopo si può usare il software *DTMF TONE DECODER* (vedi sezione *Web Link*) che usa la scheda audio del PC.

LE ROUTINE DEL PROGETTO

La routine DTMF.asm

Il file DTMF.asm è il programma principale in cui sono fatti vari settaggi, richiamate le varie routine e definito il ciclo di generazione dei toni.

La routine DTMF_DDS.asm

Il file DTMF_DDS.asm è una routine scritta per il PIC16F84A (ma estendibile ad altri della stessa famiglia) che implementa un doppio generatore DDS a 16 bit. Questa routine può essere utilizzata in altri progetti laddove si richiede la generazione di toni DTMF.
Le variabile d'ingresso della routine sono la riga e colonna relative al tasto premuto: *ROW* e *COL*.
Di seguito le varie sezioni della routine saranno esaminate ai fini didattici per una migliore comprensione del progetto.

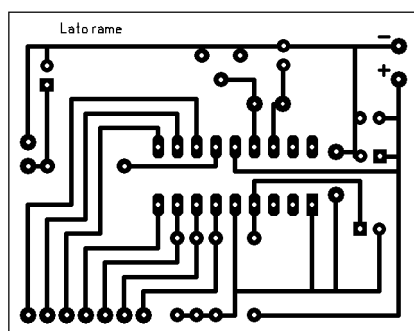



Figura 2 *Circuito stampato del generatore DTMF (lato rame)*

Figura 3 *Circuito stampato del generatore DTMF (lato componenti)*

Nella prima sezione *"DTMF_DDS: DEFINIZIONI E FORMULE"* sono riportate le formule necessarie per il calcolo delle parole di controllo ovvero *FCW1* e *FCW2* (Frequency Control Word) a 16 bit in funzione frequenze d'uscita *F1* e *F2*, componenti il tono (vedi tabella 1), in funzione del valore del quarzo usato *XTAL* e del numero di cicli macchina per 1 ciclo completo di generazione.
Le formule sono (vedi speciale sul DDS):

- F1 = FCW1 * CLK / 2^N
- F2 = FCW2 * CLK / 2^N
- CLK = XTAL/(4*Ncycle)

Essendo *FCW1*, *FCW2* < 32768 = 2^16/2 (per il teorema di Nyquist)
Per ogni tasto bisogna quindi calcolare la coppia *FCW1* e *FCW2* in funzione del quarzo. A tal scopo un foglio di calcolo (*table_calc.xls*) è stato realizzato per il calcolo della tabella di corrispondenza utilizzata per la generazione.
Nella sezione "VARIABILI: DEFINIZIONI" sono definite le variabili usate nel programma e la loro posizione nella RAM. Essendo le grandezze in gioco a 16 bit necessita definirle mediante due byte, quindi possiamo definire quanto segue nella tabella 2.

Elenco componenti

Sigla	Valore
R1, R2, R3, R4	10 KΩ 1/4 W
R5	1 KΩ 1/4 W
C1	100 nF poliestere
C2	100 µF 16 V elettrolitico
C3	10 pF ceramico
C4	4-20 pF compensatore
C5, C6	1 µF 16 V elettrolitico
T1	TASTIERA 4righe 3 colonne
U1	PIC16F84A
XTAL	8 MHz (*)

(*) La scelta del quarzo è estremamente importante.
Il valore della frequenza *XTAL* rientra nel calcolo d'alcuni parametri delle routine di generazione (per la definizione delle tabelle di corrispondenza usate nella generazione DDS (vedi il file *Table_Calc.xls*).

Nella sezione *"COSTANTI: DEFINIZIONI"* sono definite le costanti utilizzate nella routine. Precisamente è definito il valore delle costanti *H_PULSE_TIME* e *L_PULSE_TIME* che rappresentano i valori che devono assumere le variabili *H_TIMER* e *L_TIMER*). Per il calcolo dei valori da dare alle costanti *H_PULSE_TIME* e *L_PULSE_TIME* in funzione della durata tp del tono si usa la relazione:

$$tp = X * Ncycle * 4 / XTAL$$

da cui si ricava

$$X = tp * XTAL / 4 * Ncycle$$

Il valore di X, calcolato e convertito in esadecimale, è caricato nelle costanti:
H_PULSE_TIME e *L_PULSE_TIME*
Un esempio di come settare le costanti per definire la durata dei toni è fornito nel riquadro: "Come impostare la durata dei toni".

La sezione segnata come *"DTMF_DDS: GENERATORE"* è la parte In cui è implementato il doppio generatore DDS. Dovendo generare contemporaneamente due frequenze *F1* e *F2*, nella routine sono implementati due accumulatori *ACC1* e *ACC2* di 16 bit, uno per ogni frequenza che costituisce il tono DTMF. Com'è stato esposto nel tutorial sul DDS precedentemente pubblicato su Fare Elettronica, i bit più significati dei due accumulatori sono i generatori dei due toni a frequenza F1 e F2.
I due toni, rappresentati da onde quadre sono sommati mediante la funzione "OR" e il segnale risultante reindirizzato sul pin d'uscita (pin 7 della PORTB). Il numero dei cicli necessari per generare le due frequenze è: *Ncycle = 18*. In particolare *Ncycle* è il numero di cicli macchina che intercorrono tra un due operazioni di caricamento del risultato della "OR" nella porta B del PIC (*MOVWF PORTB*).
Questo vuol dire che il tono è generato in uscita da campioni che escono con una frequenza di generazione pari a CLK (vedi sopra per la formula) che è detta appunto frequenza di generazione (Fclk).
Nella sezione "*TABELLA PER FCW1* e *FCW2*" sono definite le tabelle di corrispondenza che

FCW1		FCW2		ACC1		ACC2		TIMER	
H_FCW1	L_FCW1	H_FCW2	L_FCW2	H_ACC1	L_ACC1	H_ACC2	L_ACC2	H_TIMER	L_TIMER

Tabella 2 *Variabili usate nella routine*

legano le variabili ROW e COL con i valori delle variabili H_FCW1, L_FCW1, H_FCW2, L_FCW2. Il valore riportati nelle tabelle sono calcolate in funzione d'alcuni parametri: Xtal, Ncycle, N (Numero bit).

La routine Tast.asm

Con questa routine si gestisce una tastiera 4 righe x 3 colonne mediante un processore PIC. Nel file *Tast.asm* sono riportate tutte le informazioni necessarie per una completa comprensione del suo funzionamento. La tastiera di riferimento è definita nella tabella 3.

	COL1	COL2	COL3
ROW4	1	2	3
ROW3	4	5	6
ROW2	7	8	9
ROW1	*	0	#

Tabella 3 *Tastiera di riferimento*

IL SOFTWARE DEL PROGETTO

Tra i file forniti per la realizzazione del progetto, scaricabili da sito di Fare Elettronica, è presente un foglio di calcolo *Table_calc.xls* che permette di ricavare le tabelle di corrispondenza da inserire nella routine *DTMF_DSS.asm*. Fissati i valori del quarzo Xtal e il numero di bit N (solitamente N=16) e calcolato il parametro *Ncycle* (se non sono apportate modifiche alla routine è pari a 18) il foglio di calcolo fornisce (sezione in giallo) la parte di programma da copiare ed incollare nel file DTMF_DDS.asm nella parte relativa alle tabelle ovvero nella sezione *TABELLE PER FCW1* e *FCW2*.

WEB LINK

Il decoder *DTMF TONE DECODER* necessario per il collaudo del circuito presentato in questo articolo, è prelevabile dal sito: *www.audio-software.com*. Il software relativo al progetto assieme ai files .asm e .hex sono prelevabili dal sito di Fare elettronica.

CONCLUSIONI

In quest'articolo, un generatore di toni DTMF è stato presentato come primo esempio d'implementazione della tecnica DDS esposta nello speciale pubblicato nel numero 240 (Giugno 2005) di Fare Elettronica.
Seguiranno altri progetti che implementano la tecnica DDS (riproduttore di suonerie, generatore di frequenza, ecc.).

Come impostare la durata dei toni

Si vuole una durata per i toni di 50ms, fissati *XTAL =8 MHz* e *Ncycle =18*:

X = 50E-3 * 8E6 / (4*19) = 5263d

Convertendo il valore ottenuto in esadecimale (hex) si ottiene:

TIMER = 148F h => H_TIMER=14 h e L_TIMER = 8F h

Dovranno quindi essere impostate le costanti:

H_PULSE_TIME EQU 14
L_PULSE_TIME EQU 8F

Linux nei Sistemi Embedded

In questo speciale si indicheranno le opportunità, i vantaggi e i limiti nell'utilizzo di Linux per i sistemi embedded e real-time. Oggi il sistema operativo del Pinguino sta diventando sempre più un'importante realtà nell'ambito dell'automazione industriale, questo ci ha spinto ad analizzare da un punto di vista tecnico le caratteristiche di questo S.O., per capirne i limiti e le variazioni che sono state introdotte per far fronte alle necessità del controllo e di tutte quelle attività che rientrano nelle applicazioni industriali.

INTRODUZIONE AI SISTEMI OPERATIVI PER IL CONTROLLO E LE APPLICAZIONI INDUSTRIALI

Si consideri un'applicazione di controllo con requisiti di tipo real-time, nel caso tale applicazione sia molto semplice (per esempio una centralina di controllo di un motore a scoppio) il modo in cui i vari compiti vengono eseguiti in parallelo e in cui sono utilizzate le risorse del sistema è determinato totalmente dal programma scritto dall'utente. In tutti gli altri casi, il dispositivo di controllo dovrà prevedere un minimo di sistema operativo, che si occupi almeno della pianificazione dell'esecuzione dei processi (scheduling) e della gestione della comunicazione tra i processi, quindi il requisito del tempo reale si tradurrà in caratteristiche che il sistema operativo dovrà possedere. Veniamo, quindi, a definire le proprietà principali di un sistema operativo con caratteristiche che permettano di qualificarlo come in tempo reale. Tale S.O. deve:

- Avere una politica di pianificazione dell'esecuzione che preveda un meccanismo di assegnazione di priorità ai processi, in modo che quelli a cui viene assegnata una certa priorità non siano interrompibili da altri a priorità inferiore.
- Essere multitasking pre-emptive, cioè deve essere sempre possibile interrompere un processo per trasferire le risorse a un altro che ne ha maggiormente bisogno.
- Evitare situazioni di deadlock tra processi attraverso un meccanismo di acquisizione della priorità, tali situazioni di stallo sono legate ad attese cicliche tra processi. Esistono tutta una serie di discipline da poter implementare nella gestione dei processi che evitano il verificarsi di condizioni di questo genere.
- Realizzare un meccanismo di sincronizzazione e comunicazione tra processi che sia prevedibile, in modo che essi possano scambiarsi dati in tempi certi.
- Avere noti, almeno in termini di requisiti minimi, i tempi necessari al sistema per gestire i processi, così da operare una serie di scelte con tempificazione deterministica.
- Gestire in maniera non brusca gli eventuali malfunzionamenti.

Un'altra caratteristica molto utile, di cui dovrebbe godere un buon sistema operativo per applicazioni industriali è la scalabilità: per molte applicazioni di controllo non servono tutte le funzioni che normalmente un sistema operativo svolge, per esempio per la centralina di controllo di un motore a scoppio, non c'è bisogno di un file system o della gestione di periferiche gra-

e Real-Time

di Enrico Raffone
enrico.raffone@tin.it

fiche, quindi è importante poter scegliere le sole funzionalità necessarie, in modo da non appesantire inutilmente il dispositivo di controllo (in termini di memoria necessaria e di velocità nell'esecuzione di programmi utente).

Figura 1 *Logo LynxOS*

SISTEMI OPERATIVI NELLE APPLICAZIONI EMBEDDED

Un sistema embedded è caratterizzato dal fatto di essere inserito in un dispositivo con uno scopo specifico e predefinito. Molte volte si accomuna i sistemi embedded con i sistemi real-time. Un sistema real-time è caratterizzato dall'essere dimensionato in modo da essere in grado di garantire che un ben determinato compito venga eseguito entro un certo tempo. Quanto tempo non è un parametro obbligatorio, nel senso che il requisito di essere real-time è un sinonimo di tempo deterministico e non di prestazioni. Quindi si intuisce che un sistema embedded può essere anche di tipo real-time, ma i due concetti spesso non sono richiesti implementati contemporaneamente.

SISTEMI OPERATIVI OPEN SOURCE E PROPRIETARI

Nel mondo embedded la scelta circa l'utilizzo di un sistema operativo risulta legata ai requisiti del problema da risolvere con il sistema da progettare, il caso estremo di assenza del S.O. di solito si presenta se il sistema da progettare risulta vincolato da requisiti molto stringenti in relazione ai tempi (in tal caso si parla di "hard embedded"), ciò avviene tipicamente nel caso di sistemi che non necessitano di una particolare concorrenza nell'attribuzione delle risorse, che sono tra l'altro molto limitate oppure avviene in sistemi che devono essere particolarmente ottimizzati. In ogni caso in quei sistemi dove risulta utile la presenza di un sistema operativo, gli orientamenti principali sono due, i sistemi operativi open source come Linux, e i "tradizionali" sistemi operativi proprietari.

In passato si utilizzavano soprattutto sistemi proprietari sviluppati in proprio sulla base di competenze (skill) presenti nell'azienda costruttrice, oppure si utilizzavano sistemi commerciali, come per esempio VxWorks, Neutrino, LynxOS, Integrity, uC/OS-II, ecc. specificamente progettati per sistemi dalle risorse limitate in termini di memoria, o assenza di memoria di massa ecc.

Ancora oggi può risultare valida la scelta di tali S.O., le motivazioni di tale scelta sono sicuramente legate al fatto che si riescono a ottenere con una certa facilità prestazioni in tempo reale anche molto spinte, una migliore ottimizzazione degli applicativi e soprattutto la possibilità di usufruire del fatto, che tali S.O. sono certificati per applicazioni in ambiente aerospaziale, difesa o semplicemente la possibilità di avere certificazione POSIX (Portable Operative System Interface), tale certificazione legata allo standard definito dall'IEEE e riconosciuto a livello mondiale, assicura la portabilità

Figura 2 *Una schermata di LynxOS*

del codice tra i sistemi e molto spesso è richiesta nei contratti di fornitura soprattutto dalle istituzioni governative.

Negli ultimi anni si sta diffondendo sempre di più l'utilizzo di sistemi operativi open source, tali sistemi stanno facendo molta strada perfino in ambito governativo, ad esempio anche presso il Pentagono; il comitato tecnico del progetto "Future Combat System" dell'esercito degli Stati Uniti, ha scelto Linux come S.O. di riferimento. Il successo dell'Open Source è legato al fatto che un sistema operativo di questo genere si mostra più economico in quanto privo di royalties, flessibile nell'utilizzo in quanto risulta semplice "ritagliare" dal tutto solo quello che serve per la propria applicazione embedded, in più sistemi operativi open source come Linux aggiungono anche doti di robustezza, affidabilità, flessibilità, bassi costi, possibilità di personalizzazioni ed interventi a basso livello sul codice, insomma un po' tutte quelle che sono le caratteristiche più importanti richieste da un'applicazione industriale.
Ma molto importante risulta talvolta capire cosa un determinato codice esegue e quindi prendere spunto per soluzioni personali. Al di là dell'abbattimento dei costi per le royalties, Linux è abbastanza diffuso e conosciuto da un grosso numero di sviluppatori "Linux-literate".

Figura 3 *Schermata di uClinux*

Figura 4 *Schermata di Klinux*

Figura 4b Logo Koan Software

Numerose comunità online sono disponibili a fornire un bacino di potenziali sviluppatori, che forniscono documentazione e supporto tecnico gratuito.
Quindi per le diverse aziende che scelgono l'open source è certamente più facile trovare un ingegnere con qualche conoscenza di sviluppo sotto Linux, che uno che conosca ad esempio LynxOS. Infine c'è anche la possibilità di usufruire di vari fornitori in termini di diverse distribuzioni, che propongono comunque lo stesso kernel Linux. Quindi un pluralismo di produttori di distribuzioni Linux embedded che propongono diversi servizi e diverse offerte.
A proposito di distribuzioni Linux per sistemi embedded c'è da segnalare la distribuzione uCLinux *(www.uClinux.org)*, che ha avuto grande diffusione per le doti di leggerezza e la possibilità di funzionare anche su microcontrollori sprovvisti di MMU, tali caratteristiche sono state inserite come poi vedremo nel kernel 2.6 di Linux.
Altra distribuzione interessante è Klinux *(www.koansoftware.com/it/prd_embe_klinux.htm e www.klinux.org)* della società italiana Koan Software di Bergamo. Klinux è basato sul kernel linux-2.4.26 e supporta tutti i processori delle famiglie x86, ARM (StrongArm, XScale), è inoltre disponibile a richiesta il supporto per altri processori come SH, MIPS e VRx. È in fase di sviluppo la nuova versione basata su kernel 2.6.x., Klinux è stato impiegato con successo nei settori dell'automazione industriale, nei sistemi di visione, nei sistemi di diagnosi, nel settore automotive e diagnosi motore, della connettività remota sia su sistemi PC tradizionali che embedded con minimi requisiti di sistema.
Concludiamo questa sezione indicando una nuova tendenza dei maggiori produttori di sistemi operativi proprietari che si sono mossi evolvendo verso il mondo Linux, fornendo, quindi, accanto al loro S.O. principale, anche sistemi operativi basati su Linux. Su questa linea di evoluzione si pone l'accordo tra Wind River e Red Hat, che tempo fa ha fatto grosso scalpore, ricordiamo che Wind River è la società produt-

Figura 5 Logo BlueCat Linux di LynxWorks

trice del famoso VxWorks, mentre Red Hat Inc. è una delle società più conosciute del mondo di Linux, dando origine allo sviluppo comune della distribuzione Linux: Red Hat Embedded Linux.
Un altro esempio è LynuxWorks che accanto al blasonato LynxOS, fornisce anche la distribuzione Blue Cat Linux *(www.lynuxworks.com)*.

OPEN SOURCE - CODICE LIBERO

Una caratteristica che invita molto ad utilizzare Linux come piattaforma embedded è, come già sottolineato più volte, il fatto di essere un sistema operativo "open source". Cerchiamo di chiarire proprio questo ultimo concetto, con tale definizione ci si riferisce al fatto che si tratta di software "libero", cioè tale software non è di proprietà esclusiva di qualcuno e può essere pertanto modificato attraverso l'accesso al codice sorgente.
L'utente ha la possibilità di studiare, manipolare, migliorare il software secondo le proprie necessità, ma anche eseguirlo e distribuirlo a chiunque e ovunque per estenderne i vantaggi anche a singoli o gruppi. Tutto questo senza pagare alcuna concessione o farsi carico dell'onere di licenze "runtime" per singola macchina o processore, normalmente previste per l'uso di sistemi operativi proprietari.
È importante che il sistema operativo open source includa il codice sorgente, perché può capitare che in talune distribuzioni questo non sia compreso. In tal caso deve essere indicato con chiarezza come ottenere il codice sorgente, ad esempio attraverso lo scaricamento da un sito Internet.

Figura 6 BlueCat Linux di LynxWorks

Per avere una descrizione sicuramente più autorevole e più precisa del concetto di open source e software libero basta visitare il sito web *www.gnu.org/philosophy/free-sw.it.html*

ANALISI TECNICA DELL'UTILIZZO DI LINUX NEL MONDO EMBEDDED

Che cos'è Linux?

Questa è una domanda che necessiterebbe di una risposta molto articolata, cosa che sicuramente va oltre i limiti di spazio a noi riservati, in ogni caso cerchiamo di formulare una risposta che sia generale e comunque completa.
Linux è un sistema operativo multitasking e multiutente, disponibile per diverse piattaforme hardware tra cui anche i processori Intel e AMD. Linux appartiene alla famiglia dei sistemi UNIX (come Solaris, AIX, HPUX, SCO, etc.) ma è stato scritto per essere compatibile con le specifiche POSIX, in più include estensioni proveniente dai sistemi System V e BSD.
Linux in se stesso è costituito solo dal Kernel, il nucleo centrale del sistema operativo che controlla il funzionamento di tutto il computer. La maggior parte delle applicazioni di contorno al sistema sono sviluppate dalla GNU.
Il kernel di Linux è stato ideato da Linus Torvalds, uno studente finlandese. Attualmente contribuiscono allo sviluppo di Linux migliaia di programmatori sparsi in tutto il mondo. Tutto il lavoro viene coordinato tramite l'uso di internet.
I sorgenti del kernel sono disponibili in rete sia nella versione stabile (che termina con un numero pari), che nella versione di sviluppo (che termina con numero dispari).
Il primo rilascio del kernel è avvenuto nel 1991. La prima release del kernel della serie 2.2.x è stato rilasciata agli inizi di febbraio del 1999. Il kernel di Linux è funzionalmente equivalente a quello di Unix, quindi molto più complesso rispetto al nucleo della maggior parte dei sistemi operativi nati per applicazioni specifiche, creati, quindi, per sistemi con limitate risorse sia di calcolo sia di memoria. Le differenze principali tra i S.O. tradizionali e Linux risiedono principalmente nel fatto che i servizi offerti dal nucleo sono forniti attraverso delle chiamate di sistema realizzate attraverso interruzioni software. I driver di periferica sono parte del codice del kernel e sono accessibili attraverso il file system. In più

il codice del nucleo funziona sempre in modalità supervisore, con la possibilità di sfruttare tutte le risorse dell'hardware a disposizione, mentre il codice dell'applicazione viene eseguito in modalità utente e non può usufruire di tutte le risorse e della possibilità di gestire interruzioni ed eccezioni come avviene, invece, per il codice del nucleo. Linux per funzionare ha bisogno sempre di un file system, ma ciò non implica la necessità di un'unità di memorizzazione di massa, il file system può essere basato anche su memoria Flash oppure RAM, ma anche un file system di rete, quindi di un'altra macchina.
Circa la gestione dei processi Linux, come Unix, non prevede l'esistenza dei threats a livello di sistema, ma per poter generare un processo che esegua un programma, viene effettuata un'operazione di "fork", che a partire da un processo clona un nuovo processo, con la conseguente sostituzione dell'immagine con quella del programma da eseguire. La programmazione con i threats è comunque disponibile a livello utente.
Concludiamo sottolineando la notevole dotazione di meccanismi di comunicazione tra processi, abbiamo strutture semplici come i segnali fino ad arrivare a meccanismi come semafori, code di messaggi, monitor, ma anche meccanismi a memoria condivisa, socket, ecc.

Le applicazioni in tempo reale e la schedulazione sotto Linux

Linux non nasce per il tempo reale "stretto" ma è un sistema operativo multitasking e multiutente quindi nato più per applicazioni che richiedono requisiti di tempo meno stringenti.
Per poter capire le problematiche di gestire processi di tipo real-time dobbiamo conoscere le politiche e gli algoritmi di scheduling disponibili in Linux. Ricordiamo due concetti importanti, il primo, che lo scheduler è il componente del sistema operativo che decide istante per istante quale sia il processo che debba avere il controllo della "risorsa" CPU e il secondo, che ad ogni processo vengono assegnati due attributi fondamentali: una priorità e un algoritmo di scheduling.
Possiamo definire lo scheduling sotto Linux a partire dalla politica di scheduling, tale politica si propone il raggiungimento dei seguenti obiettivi, alcuni dei quali contrastanti tra loro:

- Timesharing.
- Gestione di priorità dinamiche.
- Avvantaggiare processi I/O-bound.
- Tempi di risposta bassi.
- Throughput elevato per i processi in background.
- Evitare attese indefinite.

Distinguendo tra politica ed algoritmo di scheduling, gli obiettivi della politica, appena elencati, sono perseguiti dall'algoritmo di scheduling nel modo seguente:

- Suddivisione del tempo in epoche.
- Dipendenza della priorità dal residuo del quanto di tempo dall'epoca precedente.
- Definizione delle condizioni che determinano l'invocazione dello scheduler.

Cerchiamo di chiarire queste affermazioni. In Linux lo scheduling si basa sul concetto di time-sharing, per cui ad ogni processo è assegnato un *quanto di tempo massimo per l'esecuzione*. La selezione del prossimo processo da eseguire è basata sul concetto di *priorità*; questa può essere *dinamica* o *statica*. In particolare, la prima è introdotta per evitare il fenomeno della starvation (attesa indefinita di un processo), mentre la seconda è stata introdotta per consentire la gestione di processi real-time. Ai processi real-time (più precisamente soft real-time, utili ad esempio per riprodurre flussi audio e/o video in applicazioni multimediali) è assegnata una priorità maggiore di quella assegnata ai processi ordinari. Di norma Linux prevede uno scheduling con prelazione (preemptive), per cui ad un processo viene tolta la CPU se:

- Esaurisce il quanto di tempo a sua disposizione.
- Un processo a priorità più alta è pronto per l'esecuzione.

Definito il quanto di tempo e la priorità passiamo alle "epoche". Il tempo di utilizzo della CPU è suddiviso in periodi, detti epoche, che si ripetono ciclicamente. All'inizio di ogni epoca, a ciascun processo viene assegnato un quanto di tempo (il valore di default è di 20 clock tick corrispondenti a 210 millisecondi). Il quanto di tempo definisce un limite superiore al tempo di utilizzo della CPU nell'epoca. Comunque, in una stessa epoca, la CPU può essere assegnata ad un processo più volte, fino a quando il quanto non è esaurito. Un'epoca termina quando tutti i processi eseguibili hanno esaurito il loro quanto. Terminata un'epoca, ne inizia un'altra; all'inizio della nuova epoca viene calcolato il nuovo quanto di tempo da assegnare a ciascun processo.
Questo è determinato come la somma tra il quanto base di tempo (*base time quantum* cioè, il valore predefinito del quanto di tempo) e metà dei clock tick eventualmente rimasti dall'epoca precedente. Infatti, processi che erano bloccati al momento in cui è terminata l'epoca, potevano non aver esaurito il quanto di tempo. Così facendo si assegna un bonus ai processi I/O-bound; creando uno squilibrio verso i processi I/O-bound, perché un processo I/O-bound difficilmente esaurirà l'intero quanto a sua disposizione. In questo modo, il numero di clock tick di un processo che non ne consuma alcuni entro un'epoca, può tendere asintoticamente a 40.
Quanto descritto è quello che in generale vale per i processi ordinari, cioè quelli che vengono gestiti con la politica detta SCHED_OTHER, a cui è assegnata una priorità statica pari a 0; mentre la priorità dinamica coincide con il numero di clock tick. Per i processi real-time non vale la suddivisione del tempo in epoche. Ai processi real-time è assegnato un livello di priorità statica compreso tra 1 e 99, che non cambia durante l'esecuzione.
Come conseguenza, i processi real-time hanno una priorità statica sempre superiore a quella dei processi ordinari. Dato che lo scheduler seleziona il processo a priorità più alta, un processo ordinario può essere eseguito solo se non ci sono processi real-time pronti per l'esecuzione.
I processi real-time possono appartenere ad una di due categorie:

- SCHED_FIFO: processi real-time con quanto di tempo illimitato. Questi processi lasciano la CPU solo se:
  - Si bloccano
  - Terminano
  - Un processo a priorità più alta passa nello stato di pronto

- SCHED_RR: processi real-time soggetti a scheduling di tipo Round Robin.

I principali problemi dell'algoritmo di scheduling di Linux sono che le prestazioni dello scheduler degradano al crescere del numero di processi. Infatti, le priorità (cioè i quanti) devono essere ricalcolate per tutti i processi al termine di ogni epoca; poi il valore di tempo predefinito associato al quanto può essere eccessivo nei sistemi con carico elevato; il supporto per i processi real-time è limitato.
L'algoritmo di scheduling di Linux, a partire dalla versione 2.6 del kernel, è stato in parte modificato (in particolare, per quanto attiene la gestione delle priorità e dei processi real-time) al fine di risolvere alcuni dei problemi indicati.

Descritti i meccanismi principali dello scheduling di Linux, è importante sottolineare che:

- Linux è un sistema preemptive per quanto riguarda i processi: questo significa che ad un processo può essere levato il controllo, secondo quanto descritto sopra, anche se il processo stesso non lo richiede esplicitamente.
- I processi di sistema, quindi quelli del kernel non possono essere sottoposti a prerilascio (preemption), in quanto lo scheduler può funzionare solo in certi momenti predefiniti oppure quando il kernel volontariamente ne invoca l'esecuzione.
  Questo significa che anche un processo "real-time" deve attendere che lo scheduler gli assegni la CPU e questo può avvenire con tempi massimi *(deadline)* non predicibili ("soft" real-time).

In particolare lo schedulatore Linux può intervenire solo in tre precise situazioni:

1. Al ritorno da interrupt.
2. Al ritorno dalle chiamate di sistema.
3. In seguito a chiamate interne al kernel.

La conseguenza principale di questo è che se c'è un processo real-time e qualche periferica richiede continuamente attenzione con una serie di richieste di interruzione che fanno scattare routine di servizio del sistema operativo, lo scheduler non può intervenire fin quanto il processo di sistema non completa e tutti i requisiti di tempo del processo real-time possono risultare di conseguenza non soddisfatti.

Le due soluzioni al problema della schedulazione

Nell'evoluzione di Linux per applicazioni embedded si è risolto il problema dello scheduling secondo due filoni separati, il primo prevede la modifica del kernel in modo da risolvere aumentando le possibilità di cambi di contesto al di là delle tre situazioni delineate nel paragrafo precedente, mentre il secondo filone prevede l'introduzione di un secondo kernel adatto al tempo reale, che veda Linux come attività meno prioritaria.
Secondo la prima soluzione abbiamo due tipi di modifiche (patch) praticate al kernel di Linux, chiamate rispettivamente "preemption patch" e "low-latency patch", la prima patch allarga l'insieme di possibili situazioni in cui il cambio di contesto è possibile, mentre la seconda segmenta i cammini troppo lunghi del codice del kernel, introducendo nell'ambito del codice del kernel chiamate esplicite allo scheduler.
In seguito tali tecniche si sono fuse, ma in ogni caso risultano soltanto poter lenire il problema senza risolverlo.
Il secondo filone è rappresentato dal progetto RTAI *(www.rtai.org)*.
Tale progetto presenta un componente software, Adeos che si occupa di fare una selezione tra eventi significativi per poi passarli al kernel RTAI o al kernel Linux.
Questo filtraggio permette di separare effettivamente le componenti effettivamente in tempo reale da quelle non real-time così da ottenere il rispetto dei requisiti.

Le novità del kernel 2.6

Il kernel Linux 2.6 realizza miglioramenti che lo rendono molto più valido per applicazioni in tempo reale:

- Sono stati inseriti dei punti di "preemption" nel kernel, cioè punti nei quali può girare lo scheduler.
- È stato ottimizzato lo scheduler, implementando algoritmi più efficienti in modo da

ridurre l'overhead introdotto dal S.O., soprattutto in presenza di molti "task".

- Il sistema può girare anche su sistemi privi di memoria virtuale (allargando di molto la base di processori supportati, cioè tutti quelli sprovvisti di MMU).
- Sono state introdotte nuove primitive di sincronizzazione, che permettono di usare i "mutex" senza ricorrere a chiamate di sistema.
- Sono stati introdotti nel kernel il supporto per le segnalazioni e per i timer ad alta risoluzione POSIX (oltre ad una ottimizzazione del meccanismo dei thread POSIX).

È inoltre possibile eliminare completamente il codice per gestire video, tastiera, eccetera, ove questi non risultino necessari.
Dal lato opposto è possibile inserire, in quanto supportati nativamente, sistemi di comunicazione quali USB 2.0 e Bluetooth.

Portabilità e Limiti di Applicabilità

Nato come un sistema per piattaforma x86, Linux ha subito moltissimi porting su un enorme numero di architetture differenti tra cui ARM, PowerPC, SH ecc., anche circa le applicazioni, sono disponibili distribuzioni nate per lavorare su più tipi di architetture.
In più a livello di librerie C, esistono progetti liberi nati per dare origine a librerie particolarmente poco stringenti dal punto di vista delle risorse, quindi molto adatte per il mondo embedded.
Circa i limiti di utilizzo di Linux in termini di risorse, l'aspetto più delicato è la memoria a disposizione, di solito si prende come riferimento come limite inferiore 2 Mb di memoria RAM e 2 Mb di memoria ROM o Flash.

CONCLUSIONI

Linux è sicuramente una alternativa da considerare come S.O. per un sistema embedded, in quanto:

- Ha caratteristiche di robustezza, sicurezza ed affidabilità.
- Abbatte i costi perchè privo di royalties ed è eventualmente scaricabile gratuitamente da internet.
- È abbastanza diffuso e conosciuto da un'ampia comunità di sviluppatori, grazie anche all'ampio uso che ne viene fatto per la didattica e le ricerca.
- Sono disponibili "porting" su molte CPU differenti.
- Assicura una completa manutendibilità, un completo controllo del codice da parte del progettista, permettendo di svincolarsi da produttori specifici.
- È estremamente "ritagliabile" e configurabile e quindi può essere reso relativamente piccolo, almeno rispetto alle caratteristiche e prestazioni che fornisce.
- È abbastanza rispondente agli standard POSIX.
- L'ambiente di sviluppo è disponibile gratuitamente e sono presenti tools e librerie paragonabili in molti casi a quelli commerciali.
- Sono comunque disponibili distribuzioni commerciali che forniscono piattaforme di sviluppo complete e molto potenti e possibilità di training e supporto diretto, al pari dei classici S.O. proprietari.
- Rende disponibili un grosso numero di protocolli di comunicazione, rendendolo particolarmente adatto alle applicazioni nei settori delle reti di telecomunicazione.
- Molti produttori di hardware (ed anche grosse software house) si stanno muovendo per fornire il supporto diretto di Linux per i propri prodotti.

BIBLIOGRAFIA

- P. Chiacchio – F. Basile – **Tecnologie informatiche per l'automazione** – McGraw-Hill, 2004.
- S. Piccardi - **Guida alla programmazione in linux** (GAPIL).
- A.S. Tanenbaum - **I Moderni Sistemi Operativi** - Jackson Libri, 2002.
- Davide Ciminaghi - **GNU/Linux nelle applicazioni embedded - Seminario: Linux e le sue applicazioni industriali, nei sistemi embedded e real-time**, Ordine degli Ingegneri di Bologna. Novembre 2004.
- Stefano Pozzi - **Linux e le sue applicazioni industriali** - Semin**ario: Linux e le sue applicazioni industriali, nei sistemi embedded e real-time**, Ordine degli Ingegneri di Bologna. Novembre 2004.

Leader nella progettazione e produzione di circuiti ibridi con tecnologia in film spesso

Modello	Alimentazione	Sensibilità RF/Potenza trasmissione	Frequenza (xxx)	Velocità trasmissione	Descrizione
RR30-xxx	5Vdc/2,5mA	-105dBm	300-450MHz	4,8Kbps	Coppia di moduli AM di dimensioni estremamente compatte (TX = 17,8 x 10,2mm, RX = 25,4 x 8,9mm), omologati I-ETS 300-220, idonei per applicazioni di controllo remoto
RT4-xxx	2÷14Vdc/4mA	+7dBm	303,8-433,92MHZ	4Kbps	

Modello	Alimentazione	Sensibilità RF	Frequenza (xxx)	Velocità trasmissione	Descrizione
RR18-xxx	3Vdc/70mA	-96dBm	433,92MHz	4,8Kbps	Ricevitore AM superigenerativo a basso consumo, con filtro saw in ingresso e banda stretta

Modello	Alimentazione	Potenza trasmissione	Frequenza (xxx)	Velocità trasmissione	Descrizione
RTQ4-xxx	2÷5,5Vdc/12mA	+7dBm	433,92MHz 868,35MHz 915MHz	9,6Kbps	Trasmettitori AM con oscillatore al quarzo, dimensioni compatte (17,78 x 10,16 mm) e pin out compatibile con il modello RT4

Modello	Alimentazione	Potenza trasmissione	Frequenza (xxx)	Velocità trasmissione	Descrizione
RTFQ4-xxx	2÷5,5Vdc/12mA	+7dBm	433,92MHz 868,35MHz 915MHz	9,6Kbps	Trasmettitore FM con oscillatore al quarzo, dimensioni compatte (17,78 x 10,16 mm) e pin out compatibile con il modello RT4

Modello	Alimentazione	Sensibilità RF	Frequenza	Velocità trasmissione	Descrizione
RXQ1-XXX	2.7÷5,25Vdc/12mA	-100dBm	433,92MHz 434,33MHz	20Kbps	Transceiver a 2 canali
TRXQ1-XXX	2.7÷5,25Vdc/12mA	-100dBm	433,92MHz 434,33MHz	20Kbps	Transceiver a 2 canali con encoder/decoder
RXQ2-xxx	2÷3,6Vdc	-100dBm	433,92MHz	38,4Kbps	Transceivers multicanale per trasmissione dati veloci e sicure Completo d'interfaccia RS-232
	2÷3,6Vdc	-100dBm	868,35MHz	38,4Kbps	

Maggiori informazioni sono disponibili nel sito **www.telecontrolli.com**,
campioni disponibili per vendita on-line tramite **www.elettroshop.com** nella sezione "Moduli RF".

gli appuntamenti

01-02 Ottobre 2005

RADIANT AND SILICON — Segrate (MI)

La Mostra comprende, tra le altre, le seguenti voci merceologiche: apparecchi e componenti per le telecomunicazioni, ricetrasmissioni, elettronica, informatica, videogiochi, surplus, editoria specializzata, radio d'epoca. Attualmente RADIANT, che si sviluppa su una superficie espositiva di 10.000 mq., conta più di 160 espositori ed oltre 12.000 visitatori.

LUOGO: Parco Esposizioni Novero - Segrate (MI)
ORARI: Dalle 9:00 alle 18:00
ORGANIZZATORE: Comis – Parco Esposizioni Novegro (www.parcoesposizioninovegro.it)
INGRESSO: € 8,00

08-09 Ottobre 2005

EXPO RADIO & INFORMATICA 2005 — Potenza

Ideata per consentire, agli operatori del settore, una migliore valutazione dell'offerta delle nuove tecnologie. Inoltre offre anche ad un pubblico più eterogeneo, in cerca di buone occasioni commerciali, tutta l'esperienza e la professionalità acquisita nel campo delle telecomunicazioni e dell'informatica.

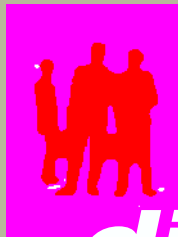
LUOGO: Quartiere Fieristico Area Industriale Tito Scalo Potenza (PZ)
ORARI: Dalle 9:00 alle 13:30 e dalle 16:30 alle 21:00
ORGANIZZATORE: EFAB srl (www.efab.it - Fax 0971.651014)
INGRESSO: n.p.

15-16 Ottobre 2005

EXPO ELETTRONICA — Faenza

Nata come mercatino radioamatoriale, nel corso degli anni l'evento si è aperto anche ad altri settori ed applicazioni, andando di pari passo con il dilagare dell'elettronica negli oggetti di uso professionale o quotidiano e radunando oltre 150 espositori provenienti da tutt'Italia. Tra i protagonisti della manifestazione computer, periferiche ed optional, decoder per la tv digitale, antenne, lettori dvd, hi-fi, video proiettori, videogiochi; articoli, ricambi, gadget e curiosità elettriche, elettroniche e digitali. Quello faentino è un appuntamento davvero imperdibile anche per il pubblico più specializzato, che qui trova materiali e componenti molto tecnici e rari da reperire, elementi indispensabili per l'autocostruzione, la personalizzare o riparare tutto ciò che è elettrico-elettronico. Grandi affari poi tra gli espositori che propongono materiali usati o "obsoleti". Alto gradimento anche per il mercatino di Radio Expò mostra scambio di apparecchi per radioamatori, radio d'epoca, militari, surplus, valvole, ricambi e riviste che si svolgerà nella sola giornata di sabato 15 ottobre, sotto la tettoia del cortile interno alla fiera e che raccoglie il consenso dei collezionisti. Saranno presenti i volontari del Coordinamento Provinciale Associazione Volontariato Protezione Civile Ravenna parteciperanno con un'esposizione mezzi e attrezzature impiegati nelle operazioni di salvataggio.

LUOGO: Faenza Fiere (Centro Fieristico Provinciale)
ORARI: Dalle 9:00 alle 18:00
ORGANIZZATORE: Blu Nautilus srl (www.blunautilus.it - Tel 0541.439573)
INGRESSO: € 7,50 (ridotto € 6,50)

ALCUNE DELLE DATE INDICATE POTREBBERO SUBIRE VARIAZIONI

Per sapere a quali appuntamenti sarà presente visita la pagina: www.farelettronica.com/fiere

19-23 Ottobre 2005

SMAU - 42ª ESPOSIZIONE INTERNAZIONALE DI ICT & CONSUMER ELECTRONICS

Milano

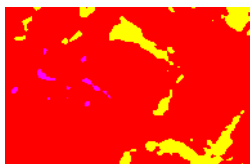
Dal 19 al 23 ottobre 2005 la Città dell'Innovazione ti aspetta a Milano.

Smau e-Business - Salone delle Tecnologie e delle Applicazioni Digitali per l'Impresa, Smau e-Government - Salone delle Tecnologie digitali al servizio del cittadino, Smau e-Life - Salone della Convergenza Digitale e della Multimedialità, IBTS - Salone Internazionale del Broadcast e delle Telecomunicazioni: quattro aree che rispondono alle esigenze di un mercato in continua evoluzione. Un grande evento espositivo che verrà arricchito da percorsi formativi e informativi "tagliati su misura", riuniti nella nuova iniziativa Smau e-Academy. Area Business (Smau e-Business, Smau e-Academy, IBTS): dal 19 al 23 ottobre - ingresso riservato esclusivamente a visitatori professionali e stampa. Area Consumer (Smau e-Government, Smau e-Life): 19 ottobre - ingresso riservato esclusivamente a visitatori professionali e stampa. Dal 20 al 23 ottobre - ingresso aperto anche al pubblico.

LUOGO: Fiera di Milano - Milano
ORARI: Dalle 9:30 alle 18:30
ORGANIZZATORE: Promotor International Spa (www.smau.it)
INGRESSO: € 10,00

29-30 Ottobre 2005

2ª MOSTRA DELL'ELETTRONICA

Scandiano (RE)

Mostra di elettronica organizzata dal Comune di Scandiano con il patrocinio della A.R.I. sezione di Scandiano. La mostra, giunta alla sua seconda edizione, è dedicata ai seguenti prodotti: telefonia, componentistica, computer, hi-fi car, radiantismo CB e OM, videoregistrazione e un mercatino delle pulci radioamatoriali.

LUOGO: Centro Fieristico di Scandiano Scandiano (RE)
ORARI: Dalle 9:00 alle 18:30
ORGANIZZATORE: Comune di Scandiano (www.fierascandiano.it - Tel 0522.857436)
INGRESSO: € 7,00 (ridotto € 4,00)

05-06 Novembre 2005

SALONE DELL'ELETTRONICA

Erba (CO)

EXPO ELETTRONICA (ex ABC dell'Elettronica) ad Erba (Como) si svolge due volte all'anno, in primavera ed in autunno. Oltre alle merceologie "tradizionali" proposte da questo tipo di manifestazioni, quali computer, elettronica in genere, radiantismo, telefonia, surplus... nonché radio d'epoca, dischi e CD da collezione.
Inoltre propone, in primavera, il Salone dell'Astronomia e Photo Cine video, con macchine ed attrezzature per la fotografia , mentre in autunno i protagonisti sono i radioamatori grazie al CB Day, a loro dedicato.
Certamente non mancano i buoni motivi per andare a dare un'occhiata; ci saranno buoni affari sia per gli esperti sia per i neofiti!

LUOGO: Centro Espositivo e Congressuale LarioFiere Erba (CO)
ORARI: Dalle 9:00 alle 18:00
ORGANIZZATORE: Blu Nautilus srl (www.blunautilus.it - Tel 0541.439573)
INGRESSO: € 7,50 (ridotto € 6,50)

Mikrobasic per PICmicro

Questa volta parliamo di suono, ovvero di come poter utilizzare il compilatore MikroBasic della MikroElektronika, in abbinamento al nostro microcontrollore PIC16F84, per produrre note, melodie e motivi musicali al fine di rendere i nostri progetti più accattivanti e professionali.

In questa puntata studiamo qualcosa di diverso dal solito. Come si legge dal titolo, parliamo appunto di musika (proprio con la lettera k) in onore del nostro Mikrobasic della mikroElektronika, che da questa puntata ci offre la nuova versione 2.0 (almeno alla data della stesura dell'articolo), piena di novità e di molte potenzialità. Invitiamo pertanto i lettori a scaricare l'ultima release dal sito omonimo.

SUONI, DOVE E QUANDO

Sono moltissime le applicazioni elettroniche che usano i suoni come feedback per l'utente: ad esempio per avvertire un operatore del verificarsi di un evento oppure per richiamare l'attenzione di un addetto al controllo di un quadro elettrico, per un problema tecnico.
Insomma, dove le periferiche di visualizzazione (display, led, lcd, lampade, ecc) non riescono a dare in pieno il loro contributo, intervengono con successo, con un ruolo insostituibile, le periferiche sonore, la maggior parte delle quali sono supportate da piccoli altoparlanti e da buzzer.
In linea di massima non si può collegare un altoparlante di bassa impedenza direttamente ad un pin del microcontrollore, ma occorre usare qualche componente in più per ottenere una potenza maggiore senza, per questo, "appesantire" il lavoro svolto dal PIC.

Suono, nota e musica

Con il termine di **suono** si intende una vibrazione meccanica che, propagandosi in un mezzo elastico, ad esempio l'aria, provoca una percezione da parte del nostro sistema uditivo. Per essere tale, la sua frequenza deve essere compresa tra 20 Hz e 20.000 Hz. (almeno per l'essere umano). A seconda della frequenza, si usa catalogare il suono in sette **note** (DO, RE, MI, FA SOL, LA, SI) distribuite tra diverse *ottave*.
Infine la **musica** è *l'arte del combinare* assieme le note al fine di produrre una sensazione piacevole di ascolto e, chiaramente, tale fenomeno è puramente soggettivo.
Non addentrandoci comunque in discorsi troppo tecnici, da un punto di vista acustico: diciamo solo che un suono è caratterizzato da una specifica forma d'onda; in altre parole la variazione che segue l'ampiezza nel tempo disegna effettivamente un grafico, di cui si possono studiarne le caratteristiche.

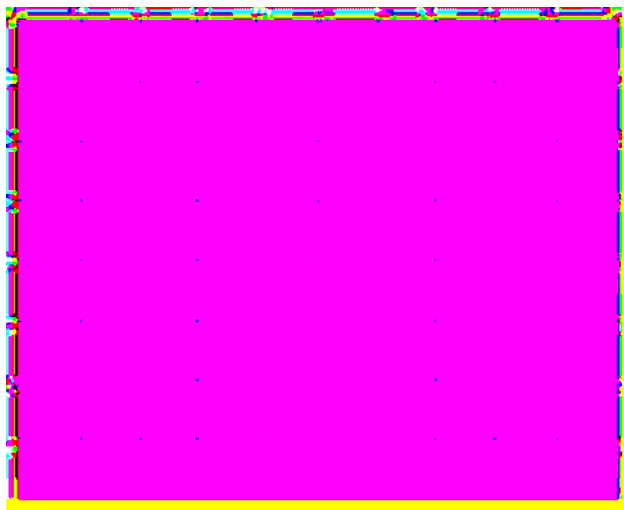
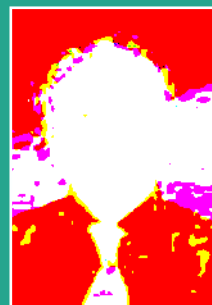



Figura 1 Tipica onda rettangolare

MUSIKA MAESTRO! Generazione di melodie, note, effetti sonori e musica con il PIC

di Giovanni Di Maria
(g.dimaria@farelettronica.com)

Dal momento che il nostro microcontrollore è capace di generare solamente due livelli di tensione (5 Volt e 0 Volt), l'unica forma d'onda che possiamo tirare fuori è **l'onda rettangolare**, ossia un treno di impulsi che, se supera una certa frequenza, genera un rumore udibile dall'uomo. Eseguiremo svariati esempi per comprendere bene tutto ciò.

T = 1 / F

Vediamo adesso, con la teoria, come sia possibile ottenere dal nostro micro, tutti i suoni udibili (e anche quelli non udibili ...).

Partiamo da un semplice esempio: occorre generare una nota di 500 Hz per la durata di un secondo. Cosa vuol dire? Semplicemente che in un "solo" secondo la sua forma più semplice si ripete completamente (dall'inizio alla fine) per 500 volte. Dal momento che la nostra nota ha una forma *puramente* rettangolare, si ripeterà nel tempo una commutazione da uno stato logico ALTO ad uno stato logico BASSO per 500 volte. Se poi vogliamo sapere quanto tempo dura un singolo evento (denominato anche periodo o ciclo) basterà dividere 1 secondo per 500 (da qui la formula T=1/F) e nel nostro caso otterremo il risultato 0,002 secondi (ossia 2 millesimi di secondo, di cui 1 millesimo in stato ON e 1 millesimo in stato OFF). Con qualche grafico il concetto viene meglio chiarito (figura 2).

Figura 2 Onda rettangolare di 500 Hz

Un'onda rettangolare di 500 Hz ha un periodo di 1/500=0,002 secondi, di cui la parte attiva dura solamente 1 ms. e altrettanto la parte "a riposo". Se non vi fosse questo alternarsi di stati non potrebbe esistere una produzione di vibrazione e quindi di suono.

	OTTAVE					
NOTA	**2**	**3**	**4**	**5**	**6**	**7**
Do	65,4	130,8	261,6	523,3	1046,5	2093
Do#	69,3	138,6	277,2	554,4	1108,7	2217,5
Re	73,4	146,8	293,7	587,3	1174,7	2349,3
Re#	77,8	155,6	311,1	622,3	1244,5	2489
Mi	82,4	164,8	329,6	659,3	1318,5	2637
Fa	87,3	174,6	349,2	698,5	1396,9	2793,8
Fa#	92,5	185	370	740	1480	2960
Sol	98	196	392	784	1568	3136
Sol#	103,8	207,7	415,3	830,6	1661,2	3322,4
La	110	220	440	880	1760	3520
La#	116,5	233,1	466,2	932,3	1864,7	3729,3
Si	123,5	246,9	493,9	987,8	1975,5	3951,1

Tabella 1 *Nomi e frequenze delle note (in Hertz)*

GENERIAMO IL NOSTRO PRIMO SUONO

Bene, siamo pronti per il primo esperimento, ossia quello della generazione di un suono di

Figura 3 Schema elettrico del prototipo sonoro

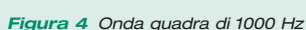
1000 Hz (1 Khz) della durata di 5 secondi. Approntiamo per lo scopo lo schema di cui alla figura 3.

Un'onda quadra di 1000 Hz (ossia di 1000 cicli al secondo) ha un periodo base di 1 ms. ossia un intero ciclo dura 1 ms. (1 / 1000).
In un periodo completo si hanno due semiperiodi: il primo di livello logico alto (5 volt) che corrisponde ad una durata di 0,5 ms. (500 microsecondi) ed il secondo di livello logico basso (0 volt) anch'esso della durata di 0,5 ms. Tale periodo, definito anche **tick**, corrisponde ad un solo ciclo di lavoro. Per ottenere una nota occorre ripeterlo tante volte nell'arco di tempo. Pertanto se il ciclo, in questo caso, dura solamente 1 millisecondo, occorre riprodurlo per 5000 volte per ottenere un suono della durata di 5 secondi. Infatti 1 ms. x 5000 volte = 5000 ms. (5 secondi).
In figura 4 possiamo vedere l'onda quadra generata con i relativi tempi di attesa.

Figura 4 Onda quadra di 1000 Hz

Andiamo adesso a scrivere il relativo listato, completo di commenti, copiando quanto riportato nel **listato 1**.

[Listato 1]

```
rem   Generazione di un SUONO da 1000 Herz
rem   della durata di 5 secondi

program suono01

dim k as word        'Indice del ciclo
dim cicli as word   'Durata in tick

main:
trisb=0    'Definisce la PORTB in uscita

portb=0    'Azzera la PORTB
cicli = 1000 * 5   'Ripetizione che assicura
                    '5 secondi
for k=1 to cicli
     portb.0=1      ' Bit ALTO
     delay_us(500)
     portb.0=0      ' Bit BASSO
     delay_us(500)
next k
end.
```

Dopo aver compilato il sorgente, scarichiamo il relativo file prodotto (con estensione HEX) all'interno del PIC e, dando alimentazione al circuito, udiremo chiaramente in altoparlante una nota di 1000 Hz della durata di 5 secondi esatti.
Se disponete di un oscilloscopio, collegandolo direttamente tra PIN RB0 e massa, potrete osservare l'oscillogramma di un'onda quadra con i tempi precedentemente descritti.
N.B. Il suono che il microcontrollore produce possiede un livello di ampiezza (volume) molto basso. Se si vuole aumentare l'intensità occorre amplificare il segnale.

SCALA MUSICALE

Il prossimo programma, peraltro molto semplice, permette l'ascolto in altoparlante di una scala musicale, formata solamente dalle note Do, Re, Mi, Fa, Sol, La, Si, Do.
Ci riferiamo sempre allo schema base riprodotto in figura 3. Ogni nota deve avere la durata di un secondo, per cui tutto il programma termina la sua esecuzione in 8 secondi.
Dal momento che ogni nota è di frequenza diversa da ogni altra, il numero dei cicli (per determinare la durata di 1 secondo) dovrà essere calcolata proporzionalmente, come nell'esempio precedente. La tabella 2 mostra i valori da inserire nel **listato 2**.

[Listato 2]

```
rem      Scala Musicale

program suono02

dim k as word

main:

trisb=0       'Definisce la PORTB in uscita
portb=0       'Azzera la PORTB
delay_ms(1000) 'Attesa iniziale di un secondo
rem -----GENERA IL DO-----
for k=1 to 262
    portb.0=1       ' Bit ALTO
    delay_us(1908) ' 3817 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1908) ' 3817 / 2
next k
rem -----GENERA IL RE-----
for k=1 to 294
    portb.0=1       ' Bit ALTO
    delay_us(1700) ' 3401 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1700) ' 3401 / 2
next k
rem -----GENERA IL MI-----
for k=1 to 330
    portb.0=1       ' Bit ALTO
    delay_us(1515) ' 3030 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1515) ' 3030 / 2
next k
rem -----GENERA IL FA-----
for k=1 to 349
    portb.0=1       ' Bit ALTO
    delay_us(1432) ' 2865 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1432) ' 2865 / 2
next k
rem -----GENERA IL SOL-----
for k=1 to 392
    portb.0=1       ' Bit ALTO
    delay_us(1275) ' 2551 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1275) ' 2551 / 2
next k
rem -----GENERA IL LA-----
for k=1 to 440
    portb.0=1       ' Bit ALTO
    delay_us(1136) ' 2273 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1136) ' 2273 / 2
next k
rem -----GENERA IL SI-----
for k=1 to 494
    portb.0=1       ' Bit ALTO
    delay_us(1012) ' 2024 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1012) ' 2024 / 2
next k
rem -----GENERA IL DO-----
for k=1 to 523
    portb.0=1       ' Bit ALTO
    delay_us(956)  ' 1912 / 2
    portb.0=0       ' Bit BASSO
    delay_us(956)  ' 1912 / 2
next k

end.
```

Nota	Frequenza (Hz)	Frequenza arrotondata	Durata di un periodo (microsecondi)
DO	261,6	262	3817
RE	293,7	294	3401
MI	329,6	330	3030
FA	349,2	349	2865
SOL	392,0	392	2551
LA	440,0	440	2273
SI	493,9	494	2024
DO	523,3	523	1912

Tabella 2 *Frequenza e durata di ogni nota*

Il programma è di una semplicità estrema. Ogni ciclo, governato dalla variabile *k*, *accende* e *spegne* la PORTB.0 collegata all'altoparlante, in modo da generare la vibrazione e quindi la nota. I cicli di attesa (delay_us) assicurano l'esatta temporizzazione tra uno stato logico alto e uno basso e tale alternanza è ripetuta sino a che la durata di una nota sia esattamente pari ad 1 secondo.

SIMULIAMO LA CORNETTA TELEFONICA

In questo prossimo esercizio realizziamo la simulazione del tono telefonico della centrale, che invita l'utente a comporre il numero da chiamare (il dial tone). In pratica dobbiamo generare una sequenza ripetitiva di toni di 425 Hz, come si vede dal flow chart rappresentato in figura 5.

Prendendo sempre come riferimento lo schema base riprodotto in figura 3, il sorgente Basic lo trovate nel **listato 3**.

Fantastico, sembra proprio di avere tra le mani un vero telefono. Commentiamo le parti salienti. Dopo aver configurato la variabile e impostata in output la PORTB, all'interno di un ciclo infinito (WHILE/WEND) sono generati i toni di cadenza e le pause.
Riprendiamo un attimo il metodo di generazione di frequenza. Si tratta di applicare "piccoli" concetti di matematica legati al tempo.

[Listato 3]

```
rem      Simulazione Cornetta Telefonica

program suono03

dim k as word     'Variabile di comodo

main:

trisb=0       'Definisce la PORTB in uscita
portb=0       'Azzera la PORTB
while true     'Ciclo infinito
   rem -----GENERA 425 Hz per 200 ms-----
   for k=1 to 85    ' (425 * 200 / 1000)
       portb.0=1       ' Bit ALTO
       delay_us(1176) ' 3817 / 2
       portb.0=0       ' Bit BASSO
       delay_us(1176) ' 3817 / 2
   next k
   rem -----GENERA pausa per 200 ms-----
   delay_ms(200)
   rem -----GENERA 425 Hz per 600 ms-----
   for k=1 to 255    ' (425 * 600 / 1000)
       portb.0=1       ' Bit ALTO
       delay_us(1176) ' 3817 / 2
       portb.0=0       ' Bit BASSO
       delay_us(1176) ' 3817 / 2
   next k
   rem ------GENERA pausa per 1000 ms----
delay_ms(1000)
wend

end.
```

Per generare, ad esempio, una nota di 425 Hz della durata di 200 ms. (1/5 di secondo) dobbiamo seguire questo iter di calcolo:

1. Calcolare il periodo (T=1/F) e quindi: **T = 1/425** ossia **T=0,002353 sec.**
2. Convertire il periodo in microsecondi, moltiplicando semplicemente il risultato per 1.000.000 (un milione) quindi: **P = 0,002353 * 1.000.000 = 2353** microsecondi.
3. Calcolare il tempo dell'onda rettangolare in ON e il tempo in OFF semplicemente dividendo per due tale periodo, quindi avremo: **T-ON e T-OFF = 2353 /2 = 1176** microsecondi.
4. I due stati logici appena calcolati devono

ripetersi per il tempo desiderato. Sapendo che, nell'esempio, un periodo completo dura solamente 0,002353 secondi, con una semplice proporzione si calcola quanti periodi occorrono per coprire il tempo necessario, quindi: **1 : 0,002353 = x : 0,2 e x = (1 * 0,2) / 0,002353 = 85** (periodi da ripetere).

Se riuscirete a far "vostra" questa procedura potrete ottenere qualsiasi frequenza e qualsiasi durata di una nota.

Esercizio per i lettori

Ora che abbiamo visto come si generano i toni, provate a programmare un simulatore di cornetta telefonica, stavolta che riproduca il tono telefonico di "linea occupata".
La nota generata è di 425 Hz con il seguente flusso: nota di 500 ms. e silenzio di 500 ms. Il tutto si ripete all'infinito.
Prima di procedere all'esercitazione consiglio "vivamente" di leggere il resto dell'articolo: potreste trovare alcuni metodi per risparmiare lavoro…

IMPARIAMO AD USARE LE PROCEDURE

Il modo di programmare (ci riferiamo anche alle precedenti puntate) è sicuramente funzionale, ma potrebbe creare qualche problema di "lunghezza del codice" nel caso si debba ripetere tante volte un certo lavoro. Ci riferiamo, ad esempio, al listato 2 delle scale musicali. Come avrete notato, all'interno di ciascun blocco FOR / NEXT è presente il codice per generare una nota. Immaginate adesso che la nostra necessità sia quella di produrre non 7 note ma molte di più. Il codice ottenuto sarebbe enorme, si potrebbe perdere facilmente il filo del discorso ma, soprattutto, la parte da copiare all'interno del PIC potrebbe non entrarvi più.
Ci vengono allora in aiuto le **procedure**, che sono come delle piccole scatole, caratterizzate da un nome (proprio come le variabili) che, a differenza, non contengono un valore numerico bensì un "pezzo" di programma, di codice da eseguire.
Con esse si possono pertanto "creare" dei nuovi comandi non presenti all'interno del compilatore. Invitiamo il lettore ad approfondire molto questo argomento poiché di importanza "vitale" ai fini della programmazione.
Ricodifichiamo nuovamente il programma della scala musicale, stavolta con una variazione: dobbiamo produrre una scala ascendente ed una discendente (DO, RE, MI, FA, SOL, LA, SI, DO, SI, LA, SOL, FA, MI, RE, DO) riferendoci sempre allo schema elettrico di figura 3. Esaminiamo il software scritto e poi passiamo ai commenti **(listato 4).**

Troviamo una grossa novità: il costrutto SUB PROCEDURE / END SUB. Come detto prima si tratta di una sorta di scatola che racchiude una parte di codice che, di solito, si deve ripetere varie volte nel programma. Utilizzando le procedure si usa dire che la programmazione diventa *modulare*.
Vediamo come esse funzionano. Innanzitutto ogni procedura deve essere "dichiarata" prima della label MAIN. All'interno di questa, possiamo scrivere tutto il codice che vogliamo ma, in ogni caso, deve trattarsi di un insieme di istruzioni generalmente "ripetitive".
Nell'esempio proposto, dopo la label MAIN, vengono elencate le note, in sequenza di riproduzione, e questa sequenza potrebbe essere completamente modificata se le esigenze di programmazione lo richiedessero.
Con le procedure abbiamo quindi creato dei nuovi comandi (Do1, re1, mi1 e così via). Basterà poi specificare quale "comando" occorre eseguire per avviare la riproduzione della singola nota.

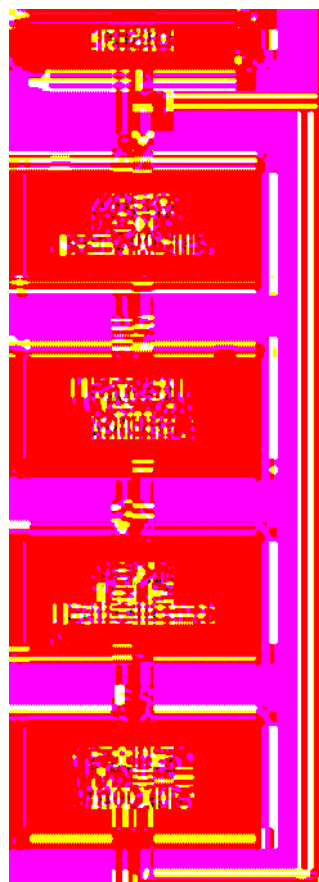



Figura 5 Flow chart del segnale di dial tone

[Listato 4]

```
rem  Scala Musicale che utilizza le PROCEDURE

program Suono04

dim k as word     'Variabile di comodo

rem -----GENERA IL DO-----
sub procedure do1
for k=1 to 262
     portb.0=1        ' Bit ALTO
     delay_us(1908) ' 3817 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1908) ' 3817 / 2
next k
end sub
rem -----GENERA IL RE-----
sub procedure re1
for k=1 to 294
     portb.0=1        ' Bit ALTO
     delay_us(1700) ' 3401 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1700) ' 3401 / 2
next k
end sub
rem -----GENERA IL MI-----
sub procedure mi1
for k=1 to 330
     portb.0=1        ' Bit ALTO
     delay_us(1515) ' 3030 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1515) ' 3030 / 2
next k
end sub
rem -----GENERA IL FA-----
sub procedure fa1
for k=1 to 349
     portb.0=1        ' Bit ALTO
     delay_us(1432) ' 2865 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1432) ' 2865 / 2
next k
end sub
rem -----GENERA IL SOL-----
sub procedure sol1
for k=1 to 392
     portb.0=1        ' Bit ALTO
     delay_us(1275) ' 2551 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1275) ' 2551 / 2
next k
end sub
rem -----GENERA IL LA-----
sub procedure la1
for k=1 to 440
     portb.0=1        ' Bit ALTO
     delay_us(1136) ' 2273 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1136) ' 2273 / 2
next k
end sub
rem -----GENERA IL SI-----
sub procedure si1
for k=1 to 494
     portb.0=1        ' Bit ALTO
     delay_us(1012) ' 2024 / 2
     portb.0=0        ' Bit BASSO
     delay_us(1012) ' 2024 / 2
next k
end sub
rem -----GENERA IL DO-----
sub procedure do2
for k=1 to 523
     portb.0=1        ' Bit ALTO
     delay_us(956)  ' 1912 / 2
     portb.0=0        ' Bit BASSO
     delay_us(956)  ' 1912 / 2
next k
end sub

rem ----DA QUI INIZIA IL PROGRAMMA
main:
trisb=0       'Definisce la PORTB in uscita
portb=0       'Azzera la PORTB
do1          ‘ Richiama la procedura DO1
re1
mi1
fa1
sol1
la1
si1
do2
si1
la1
sol1
fa1
mi1
re1
do1
end.
```

Adottando questo metodo otteniamo un codice molto leggibile e ben organizzato.

UTILIZZIAMO LE FUNZIONI BUILT-IN

Premessa

Il compilatore Mikrobasic è dotato di alcune funzioni "preimpostate" che generano, con *pochissime* righe di programmazione, qualsiasi frequenza.
Di conseguenza tutto ciò che è stato spiegato sino ad ora sarebbe teoricamente inutile: vedremo però che tali funzionalità automatiche di *libreria* hanno alcune limitazioni che ne pregiudicano il corretto utilizzo, pertanto il conoscere l'esatta procedura della generazione del suono, scrivendo in proprio il codice specifico, non solo dà una completa comprensione di quanto accade all'interno del PIC, ma contribuisce soprattutto a risolvere e superare gli ostacoli che queste funzioni di libreria potrebbero causare.

Le funzioni

Le funzioni supportate dal Mikrobasic, almeno nella versione attuale sono due:

```
Sound_init(porta,pin)
Sound_play(periodo/10,n. periodi)
```

La prima predispone il microcontrollore ad accettare su una porta (es. PORTA o PORTB) un dispositivo sonoro di output (altoparlante o buzzer). Tale dispositivo deve essere collegato sul *pin* specificato dal secondo parametro. Per esempio la funzione `Sound_init(PORTB,0)` prepara la PORTB, e precisamente il pin RB0, ala produzione di suoni.

La seconda invece si occupa della vera e propria riproduzione della nota, generando un'onda quadra di opportuna frequenza. Il suo utilizzo potrebbe essere inizialmente un po' complicato. Vediamo come si usa.
Come si vede dalla sintassi della funzione, essa accetta (tra parentesi) due parametri: il primo indica la lunghezza del periodo della nota (in microsecondi) da dividere per 10 e determina pertanto la **frequenza** da generare; il secondo indica invece il numero di periodi che devono comporre la lunghezza della nota e determina quindi la **durata** della stessa. Per esempio la funzione `Sound_play(114,3520)` serve per generare una nota di 880 Hz (LA) di 4 secondi di durata. Un po' macchinoso a dire il vero ma non possiamo avere tutto sul piatto d'argento…
Vediamo comunque come avviene il calcolo dei parametri di tale funzione.

Calcolo dei parametri

Un esempio chiarirà meglio il concetto.
Supponiamo di avere bisogno di una nota di 880 Hz (LA) che duri 4 secondi. Andiamo a calcolare i due parametri.

1. Primo parametro *(frequenza)*
   a. Calcolare il periodo con la formula T = 1 / F (1 / 880); quindi 0,00113636.
   b. Trasformare il risultato in microsecondi, moltiplicandolo semplicemente per un milione, quindi 1136,3636 us.
   c. Dividere tale risultato per dieci e arrotondare, in quanto la funzione lo esige, quindi **114**.
   d. Abbiamo quindi ottenuto il primo parametro.
2. Secondo parametro *(durata)*
   a. Dal momento che la frequenza desiderata è di 880 Hz (cicli al secondo) è ovvio che per durare un secondo tale nota ha bisogno di 880 periodi. Per ottenere tempi maggiori o minori bisogna calcolare proporzionalmente tale dato e in questo caso moltiplicare 880 per 4 (secondi) quindi **3520**.
   b. Abbiamo quindi ottenuto anche il secondo parametro.

Invocando pertanto la funzione `Sound_play(114,3520)` otteniamo un suono della frequenza di 880 Hz della durata di 4 secondi. In ogni caso, per il calcolo dei suddetti parametri, consiglio vivamente l'adozione e l'utilizzo di un programma per la gestione di fogli elettronici, soprattutto per accelerare lo svolgimento delle operazioni matematiche, quando si hanno molte note da gestire.

Limitazioni

Utilizzando le funzioni built-in, abbiamo un vantaggio e due svantaggi. Il vantaggio consiste nel fatto che con una sola riga di codice si può

generare facilmente una nota. Il primo svantaggio consiste nell'impossibilità di riprodurre "tutte" le frequenze udibili dall'uomo, infatti il *range* d'azione spazia da un minimo di 393 Hz ad un massimo di 100.000 Hz. Come vedete il limite superiore può andare benissimo, un po' meno il limite inferiore…
Il secondo svantaggio sta nel fatto che alcune note prodotte subiscono una "leggerissima" stonatura per via di sofisticati calcoli matematici eseguiti all'interno del compilatore.

Suoniamo un "giro di DO"

Adesso facciamo riprodurre al nostro PIC un giro di DO con questa sequenza: DO, MI, SOL, DO2, SOL, MI ripetendosi all'infinito. Ogni nota deve durare 250 ms. Guardando la tabella 2, abbiamo utilizzato le seguenti note:

- DO = 523 Hz
- MI = 659 Hz
- SOL = 784 Hz
- DO2 = 1046 Hz

Ed ecco il listato, peraltro cortissimo **(listato 5)**.

[Listato 5]

```
rem     IL GIRO DI DO

program suono05

main:

sound_init(PORTB,0)
while true
   sound_play(191,131)
   sound_play(152,165)
   sound_play(128,196)
   sound_play(96,261)
   sound_play(128,196)
   sound_play(152,165)
wend

end.
```

Come si nota, non abbiamo usato alcuna variabile. Dopo la definizione del PIN 0 di PORTB, al quale occorre collegare un piccolo altoparlante, all'interno di un ciclo infinito WHILE / WEND il PIC esegue sequenzialmente le istruzioni che generano le note. Semplice vero? Un'ultima nota: non è stato necessario configurare il registro TRISB in quanto la funzione SOUND_INIT se ne fa carico completamente.

FINALMENTE LA MUSICHETTA

Adesso che abbiamo tutti gli "ingredienti" e le conoscenze per produrre suoni e note, ci possiamo cimentare nell'arduo compito di programmare il microcontrollore per l'esecuzione di una famosissima melodia.

Tanti auguri a te

Il brano in questione da riprodurre è conosciuto universalmente e si tratta della canzoncina *"Tanti auguri a te"*, tradotta in tutte le lingue del mondo e utilizzata durante i compleanni.
Per poter scrivere sul PIC la procedura serve naturalmente conoscere non solo la sequenza delle note, ma anche la loro durata e soprattutto si devono programmare le eventuali pause di silenzio. Occorre pertanto una "minima" cultura di notazione musicale.

Lo spartito

In figura 6 possiamo leggere lo spartito del brano in questione; spartito che sarà "tradotto" in numeri e valori di frequenza e durata da far digerire al povero PIC.

Tabella delle note (tabelle 3)

Con un po' di pazienza andiamo a convertire le note del pentagramma in numeri, seguendo il metodo per il calcolo dei parametri sopra descritto.
N.B. Si assume che ogni **croma** sia della durata di 200 ms. (quindi un'intera battuta di 600 ms. essendo di 3/8 la divisione del tempo).

Spiegazione della tabella

Per la riproduzione del motivo musicale, nella tabella 2 abbiamo calcolato, di tutte le note, i due parametri fondamentali (nelle caselle in grassetto).
Essa è formata da 9 colonne che andiamo ad illustrare:

1. La prima colonna contiene semplicemente il testo e le parole "cadenzate e ritmate" della canzone (vedi spartito).

fare
elettronica
FIRMWARE

2. La seconda colonna contiene il nome della nota riprodotta.
3. La terza colonna contiene la frequenza di ogni nota prodotta (vedi tabella 2).
4. La quarta colonna contiene la durata di ogni nota, espressa in millisecondi (vedi spartito).
5. La quinta colonna contiene la lunghezza del periodo della nota, espressa in secondi e viene calcolata con la formula T = 1 / F.
6. La sesta colonna contiene la lunghezza del periodo, stavolta espresso in microsecondi e si ottiene moltiplicando il periodo calcolato per 1.000.000 (un milione).
7. La settima colonna contiene il valore della sesta colonna diviso 10, come richiesto dal compilatore, e costituisce il **primo parametro** da inserire nella funzione sound_play.
8. L'ottava colonna contiene il numeri di periodi che occorrono per ottenere una nota di un secondo (se notate bene corri-

Testo Canzone	Nota	Frequenza (Hz)	Durata (ms.)	Periodo (Sec.)	Periodo (Microsec.)	Periodo Us/10	Periodi per 1 secondo	Periodi Effettivi
Tan	Sol	784	400	0,001276	1275,51	**128**	784	**314**
ti au	Sol	784	200	0,001276	1275,51	**128**	784	**157**
gu	La	880	600	0,001136	1136,36	**114**	880	**528**
ri	Sol	784	600	0,001276	1275,51	**128**	784	**470**
a	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
te	Si	988	600	0,001012	1012,15	**101**	988	**593**
	Pausa		600					
Tan	Sol	784	400	0,001276	1275,51	**128**	784	**314**
ti au	Sol	784	200	0,001276	1275,51	**128**	784	**157**
gu	La	880	600	0,001136	1136,36	**114**	880	**528**
ri	Sol	784	600	0,001276	1275,51	**128**	784	**470**
a	Re2	1175	600	0,000851	851,06	**85**	1175	**705**
te	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
	Pausa		600					
Tan	Sol	784	400	0,001276	1275,51	**128**	784	**314**
ti au	Sol	784	200	0,001276	1275,51	**128**	784	**157**
gu	Sol2	1568	600	0,000638	637,76	**64**	1568	**941**
ri	Mi2	1318	600	0,000759	758,73	**76**	1318	**791**
ma	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
es	Si	988	600	0,001012	1012,15	**101**	988	**593**
tro	La	880	600	0,001136	1136,36	**114**	880	**528**
Tan	Fa2	1397	400	0,000716	715,82	**72**	1397	**559**
ti au	Fa2	1397	200	0,000716	715,82	**72**	1397	**279**
gu	Mi2	1318	600	0,000759	758,73	**76**	1318	**791**
ri	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
a	Re2	1175	600	0,000851	851,06	**85**	1175	**705**
te	Do2	1046	600	0,000956	956,02	**96**	1046	**628**

***Tabella 3** Parametri fondamentali di tutte le note che compongono la canzone*

sponde alla sua frequenza …).

9. La nona colonna infine contiene il numero dei periodi occorrenti per formare una nota di durata specificata nella quarta colonna, e la si ottiene con una semplice proporzione. Costituisce pertanto il **secondo parametro** calcolato della funzione musicale.

Andiamo a scrivere adesso il listato 6 nell'editor del compilatore e, senza commettere errori di battitura, carichiamolo sul PIC, dopo la sua compilazione.

[Listato 6]

```
rem    TANTI AUGURI A TE

program suono06

main:

sound_init(PORTB,0)  'Inizializza la RB0

rem     Il PIC Inizia a SUONARE
rem     ... e non la smette piu' .....
while true
    sound_play(128,314)
    sound_play(128,157)
    sound_play(114,528)
    sound_play(128,470)
    sound_play(96,628)
    sound_play(101,593)
    delay_ms(600) ' PAUSA DI SILENZIO
    sound_play(128,314)
    sound_play(128,157)
    sound_play(114,528)
    sound_play(128,470)
    sound_play(85,705)
    sound_play(96,628)
    delay_ms(600) ' PAUSA DI SILENZIO
    sound_play(128,314)
    sound_play(128,157)
    sound_play(64,941)
    sound_play(76,791)
    sound_play(96,628)
    sound_play(101,593)
    sound_play(114,528)
    sound_play(72,559)
    sound_play(72,279)
    sound_play(76,791)
    sound_play(96,628)
    sound_play(85,705)
    sound_play(96,628)
    delay_ms(600) ' PAUSA FINALE DI SILENZIO
wend

end.
```

È stata molto dura ma ce l'abbiamo fatta. Il nostro microcontrollore è adesso capace di riprodurre "fedelmente" la musichetta di augurio, senza sbagliare mai una sola nota e per un numero illimitato di volte.
Il listato, seppur lungo, non ha assolutamente bisogno di commenti.
Occorre solo impiegare il tempo per calcolare tutti i valori necessari ma, ripetiamo, con un software di foglio elettronico il compito sarà sicuramente agevolato.

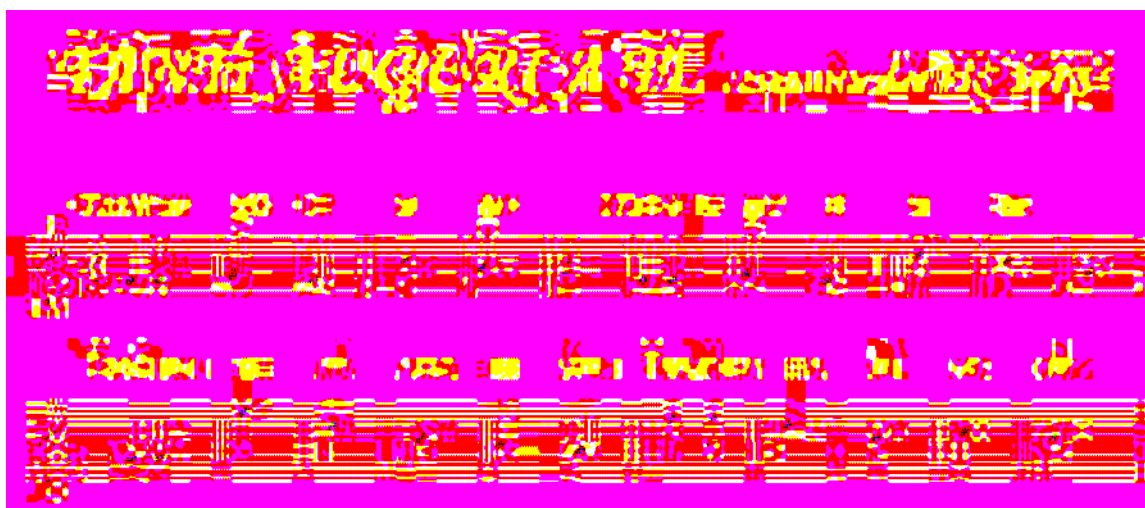



Figura 6 *Lo spartito di "Tanti auguri a te"*

PULSANTI SONORI CON BEEP

Come ultimo esempio di questo interessante articolo proponiamo una rielaborazione di un progetto presentato qualche numero fa.
Si tratta di realizzare un contatore, composto da una sola cifra (un display a 7 segmenti) che proceda avanti ed indietro di una unità, utilizzando semplicemente due pulsanti normalmente aperti. Realizziamo per lo scopo lo schema di cui alla figura 7.

Di cosa si tratta

In pratica si ha un display, composto da un visualizzatore a 7 segmenti, che può mostrare tutti i numeri da 0 a 9, semplicemente premendo uno dei due pulsanti (avanti e indietro) normalmente aperti.
Rispetto alla versione precedente però ci sono due differenze sostanziali ed elaborate:

1. Ogni qual volta che un pulsante è premuto, avviene la generazione di un "beep" di 1000 Hz.
2. Se si cerca di superare il limite raggiunto dalla cifra (ossia se ci si vuole spostare sotto lo 0 o sopra il 9) avviene la generazione di un tono lungo di 400 Hz (error tone).

Come si vede, dotando il prototipo di un'unità per la produzione di un suono, il lavoro dell'utente preposto all'utilizzo dei pulsanti è di gran lunga agevolato, in quanto ogni azione manuale (pressione del pulsante) è accompagnato da un suono che descrive l'operazione compiuta. Un esempio di quanto spiegato lo abbiamo nelle fotocopiatrici più sofisticate, dotate appunto di tastiera "sonora".

Il software

Trascriviamo il programma seguente nell'editor, compiliamo e scarichiamo il file prodotto con estensione HEX sul nostro micro **(listato 7)**.

Il programma contiene preziosismi e trucchi vari. Abbiamo definito l'array di costanti, *display*, che contiene i codici delle dieci cifre da visualizzare.
La variabile *k* invece ha il compito di tenere in memoria il conteggio.
La variabile *premuto* ha l'importante funzione di comunicare al PIC se uno dei tasti è appunto premuto o meno. Questo per evitare il fastidioso effetto del "rimbalzo". Costituisce pertanto, come segnalatore, un flag che accerta la pressione di uno dei pulsanti.
Si passa poi all'azzeramento e alla definizione delle porte attraverso i soliti registri PORT e TRIS.
Adesso viene il bello. Tutto il programma itera attorno ad un ciclo infinito, limitato dagli statements *WHILE / WEND*. Tale ciclo non termina mai

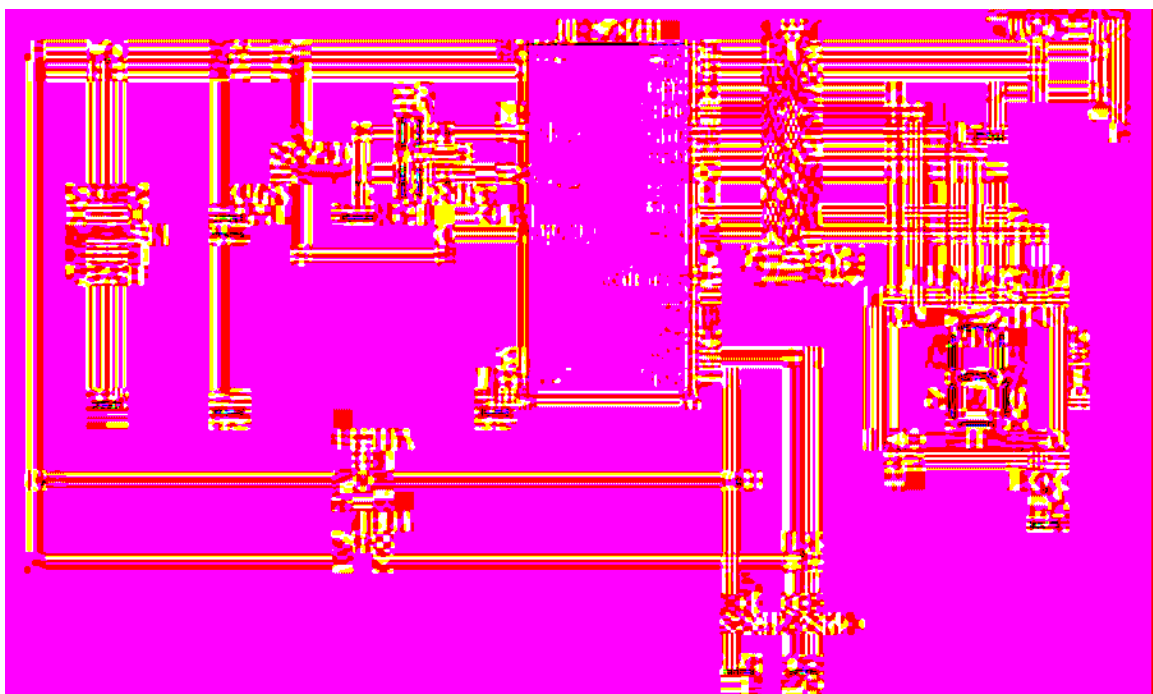



Figura 7 *Schema di contatore up-down con due pulsanti n/a con beep*

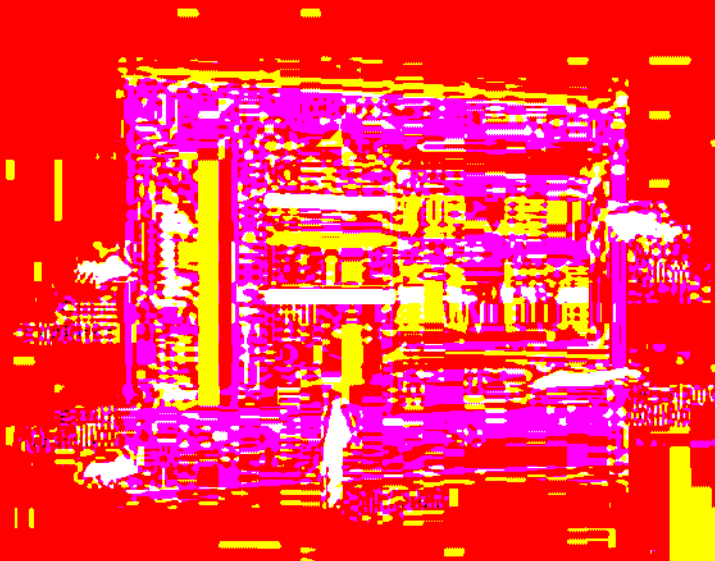
10-11 dicembre 2005
21ª Mostra Mercato Nazionale
Radiantistica Elettronica
PLAY STATION
Videogiochi
Flippers
Orario
9,00-13,00
15,00-19,30

[Listato 7]

```
rem    Display Contatore con tasti BEEP

program suono07

const display as byte[10]= (63,6,91,79,102,109,125,7,127,111)
dim k as byte 'Variabile che contiene il NUMERO (0-9)
dim premuto as byte  'Flag che indica lo stato dei tasti

main:

   porta=0 'Azzera porta
   portb=0 'Azzera portb
   trisa=3 '00011 'RA0 e RA1 Input
   trisb=0 'definisce PORTB in output

   sound_init(PORTB,7) 'Inizializza altoparlante in RB7
   k=0        'Azzera Cifra
   premuto=0  'Presuppone che nessun tasto sia premuto
   while true
      '------Processa TASTO 1-------
      if (porta.0=1) and (k<9) and (premuto=0) then 'Se premo Tasto 1
         k=k+1 'Incrementa contatore
         premuto=1  'Imposta il flag di premuto
         sound_play(100,100) '1000 Hz 100 ms
      end if
      '------Processa TASTO 2-------
      if (porta.1=1) and (k>0) and (premuto=0) then 'Se premo Tasto 2
         k=k-1 'Decrementa contatore
         premuto=1  'Imposta il flag di premuto
         sound_play(100,100) '1000 Hz 100 ms
      end if
      '---Se non si preme niente il flag si azzera--
      if (porta.0=0) and (porta.1=0) then
          premuto=0
      end if
      '---Se si tenta di andare SOTTO il limite 0---
      if (porta.1=1) and (k=0) and (premuto=0) then
         sound_play(250,400) '400 Hz 1 sec.
      end if
         '---Se si tenta di andare SOPRA il limite 9---
      if (porta.0=1) and (k=9) and (premuto=0) then
         sound_play(250,400) '400 Hz 1 sec.
      end if
      '----Visualizzazione
      portb=display[k] 'visualizza numero
      delay_ms(100) 'Pausa per ANTIRIMBALZO
   wend

end.
```

in quanto è specificata una condizione vera (true) che non potrà mai cambiare.
Con la prima istruzione *IF* viene processato il tasto su RB0 che è adibito all'incremento del numero. Per aumentare di una unità la cifra, occorre il verificarsi di **tre** condizioni, ossia:

1. Che venga premuto il tasto (porta.0=1).
2. Che il contatore non si trovi al limite del suo conteggio (k<9).
3. Che il tasto stesso non sia già premuto (per evitare l'effetto repeat).

Solo così il programma può aumentare di una unità la variabile *k* (k=k+1). Inoltre, all'interno dello stesso blocco IF/THEN, è presente l'istruzione che genera il beep, ossia un tono di 1000 Hz della durata di 100 ms. grazie alla funzione *sound_play(100,100)*.
Il tasto di decremento segue la stessa filosofia di ragionamento, con le ovvie variazioni del caso.
Il valore della variabile *k* è così visualizzato sul display attraverso il comando *portb=display[k]* che assegna alla PORTB il valore di codifica del numero. Infine riveste tantissima importanza lo stato di attesa finale, attraverso il comando *Delay_ms(100)* che elimina completamente l'effetto rimbalzo, semplicemente rallentando l'esecuzione del programma a livelli più "umani".

CONCLUSIONI

Siamo arrivato alla fine anche di questa puntata. Sarebbero ancora tantissime le cose da dire ma lo spazio a disposizione è finito. Inventate da voi gli esercizi e i prototipi più disparati per verificare praticamente ciò che abbiamo imparato e ricordate che se un programmatore scrive un programma e questo funziona subito, è stato bravo, ma se scrive un programma che non funziona e poi scopre e corregge gli errori, allora è stato doppiamente bravo!
Alle prossime puntate dunque con tanti interessanti esperimenti e studi, per utilizzare al meglio il microcontrollore PIC. Vi aspettiamo.

CPLD by Example

Inizia da oggi CPLD By Example, un corso introduttivo sulle CPLD rivolto in particolare a coloro che per la prima volta si avvicinano al mondo delle logiche programmabili, ma indirizzato anche a quanti, già esperti, desiderano approfondire la conoscenza di questi interessanti componenti.
Ognuna delle puntate è suddivisa in moduli in cui vengono affrontati aspetti circuitali e caratteristiche del linguaggio Verilog.

Prenderemo in considerazione la famiglia di CPLD Xilinx XC9500. Il corso si prefigge di mettere in grado il lettore di utilizzare questi dispositivi insieme agli strumenti di sviluppo software che sono messi a disposizione gratuitamente sul web da Xilinx Corporation.
Come anticipazione, possiamo dire che ciascun modulo tratterà un particolare argomento o aspetto della struttura interna della CPLD ed elementi sintattici del linguaggio Verilog, sempre in relazione a circuiti applicativi di esempio.

In questa prima puntata ci soffermeremo in particolare sull'evoluzione dei componenti programmabili. Vedremo poi le caratteristiche principali di alcune famiglie di CPLD e le architetture elaborate dai diversi produttori, per arrivare infine ai prodotti Xilinx che useremo nei nostri esempi.
Per motivi di spazio non sarà possibile trattare altre tipologie di componenti o di tools. Sarà comunque utile gettare le basi per ulteriori approfondimenti futuri.

Figura 1 Alcuni esemplari Xilinx

INTRODUZIONE

Quante volte ci siamo sentiti dire che il tipico personal computer dei nostri giorni è molto più potente del più grande supercomputer di solo una ventina di anni fa?
Questa è proprio la più evidente testimonianza, sotto gli occhi di tutti, dell' incredibile evoluzione che la tecnologia del computer ha avuto negli ultimi decenni. Al contrario, molto di rado si sente parlare di un altro fenomeno, meno visibile ma non meno importante: di come cioè un piccolo dispositivo programmabile e poche righe di codice possono oggi rimpiazzare letteralmente centinaia di circuiti integrati discreti.

Convenienza delle logiche programmabili

I componenti logici discreti, a lungo considerati i cavalli di battaglia dell'industria dei semiconduttori, sono stati per parecchi anni avvantaggiati dai bassi costi, rispetto alle prime logiche programmabili comparse agli inizi sul mercato.
Tuttavia, oggi le cose sono cambiate significativamente. I progressi compiuti negli anni più recenti nel campo delle tecnologie di costruzione delle CPLD hanno abbassato i costi di questi dispositivi a un livello tale da farli diventare alternative molto appetibili rispetto alle logiche fisse discrete.
Ma, al di là delle semplici ragioni di costo, vi sono altri fattori particolarmente rilevanti che rappresentano notevoli vantaggi a favore delle CPLD: la flessibilità costituita dalla riprogrammabilità, la rapidità nell' apportare le modifiche, la

Evoluzione dei componenti programmabili

di Agostino Rolando
a.rolando@farelettronica.com

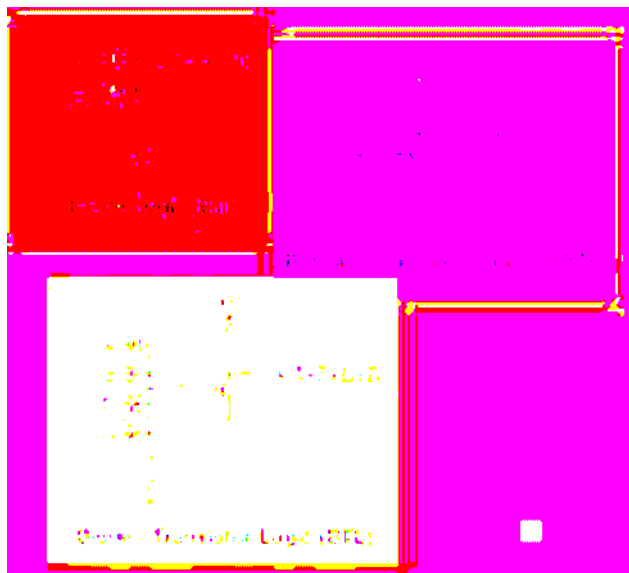



Figura 2 *Primi circuiti logici*

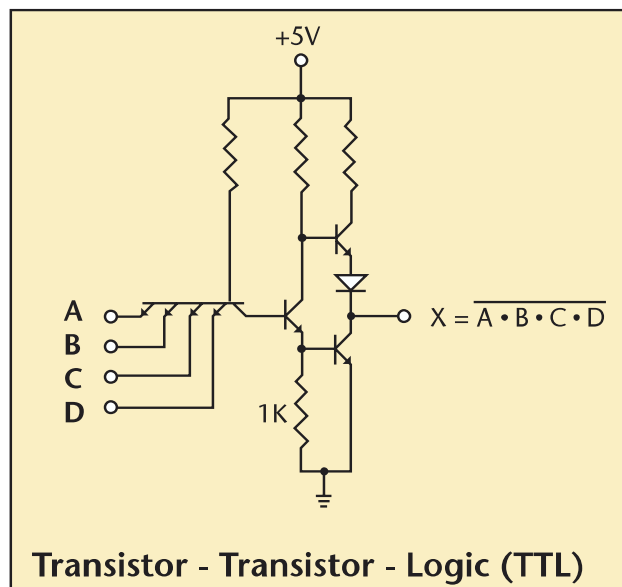
Figura 4 *Logica nand TTL*

riduzione dell'area occupata sullo stampato, la superiore affidabilità rispetto alle logiche discrete, il ridotto time-to-market, solo per citare i più importanti.

Le CPLD costituiscono oggi la soluzione ottimale per fornire pari, e in molti casi migliori, prestazioni rispetto ai più veloci tra i dispositivi logici discreti.

Figura 3 *Un esemplare della famiglia 74*

L'industria ha da tempo recepito questo messaggio, dal momento che sono sempre più numerosi i prodotti che fanno uso di CPLD, con una proporzionale diminuzione dell'impiego di integrati logici tradizionali.
Anche nel campo hobbistico l'utilizzo di questi componenti è molto promettente.

UN PO' DI STORIA

Oltre quattro decenni fa i circuiti logici digitali venivano interamente realizzati con resistenze, diodi e transistor.
A questo proposito possiamo ricordare le soluzioni a Diode Logic, Resistor-Transistor Logic, Diode-Transistor Logic (figura 2).

Ma i gate logici elementari, i contatori e le altre funzioni logiche di media complessità, si dimostrarono essere troppo ingombranti, se realizzati con componenti discreti. A causa di questi limiti, si rese necessario sviluppare la tecnologia dei circuiti integrati, per poter raggiungere gli attesi livelli di integrazione.Il risultato fu l'introduzione delle prime famiglie commerciali di dispositivi logici digitali; in Italia venivano chiamati "micrologici".
La Texas, agli inizi degli anni anni '60 commercializzò alcuni dispositivi DTL, tra cui possiamo ricordare la serie 930, a cui fece seguito la serie SN73 / 53, che rimase in produzione fino alla metà degli anni '70.
La SN53 era specifica per le applicazioni militari, con un range di temperatura operativa da -55 a +125 gradi centigradi. La serie SN73, invece, era indirizzata alle applicazioni industriali,

con un range operativo da 0 a +70 gradi. In seguito si svilupparono i circuiti integrati conosciuti come Transistor-Transistor-Logic o TTL. La classica serie 74 (54 nell'equivalente militare) fu il capostipite di quello che sarebbe diventato un'ampio set di famiglie, tutte con componenti simili nelle funzionalità, ma con significative differenze nelle performance e nei consumi individuali.

Figura 5 Partizioni sul wafer di silicio

L'era delle logiche TTL

La tecnologia delle logiche discrete TTL ottenne il battesimo ufficiale quando, agli inizi degli anni '60, la Texas Instruments introdusse la già accennata famiglia 74 di circuiti integrati *standard logic* per supportare i programmi di esplorazione spaziale, in particolare lunare, della NASA.
La famiglia 74 (figura 3) comprendeva le porte logiche elementari, come ad esempio il quadruplo NAND 7400 (figura 4), il doppio flip flop tipo D 7474, il contatore decimale 74160, i sommatori binari e altre semplici sotto-funzioni.
I primi integrati erano disponibili nei package dual-in-line a 14 e 16 pin.

I successivi progressi nel packaging e l'introduzione di dispositivi a livelli sempre crescenti di integrazione consentì ai progettisti di minimizzare drasticamente gli ingombri sui circuiti stampati e di ridurre il numero di componenti complessivo, fornendo nel contempo un comodo sistema "ad incastro" del tipo a blocchetti Lego.
Nuove famiglie venivano introdotte continuamente, atte a soddisfare le necessità di aumento delle prestazioni, minori consumi e riduzione dei costi. Il risultato fu la proliferazione di oltre trenta differenti famiglie 74, ciascuna offrente specifiche caratteristiche.

Attraverso gli anni '70, le varianti nelle famiglie TTL si diffusero rapidamente e, di pari passo, ne aumentò la complessità. Nei primi anni '80 era presente sul mercato un' estrema varietà di tipologie: TTL, S, LS, AS, F, ALS, CD4000, HC, HCT, BCT, AC, ACT, FCT, ABT, LVT, AHC, AHCT, costringendo i progettisti a destreggiarsi a fatica tra le caratteristiche di ognuna e le reciproche compatibilità elettriche, per poter al meglio realizzare le proprie applicazioni.

La nascita delle CPLD

Mentre la gamma dei dispositivi 74 si espandeva, cominciava ad apparire una nuova tendenza rivoluzionaria nella forma dei componenti logici programmabili, detti PLD (*Programmable Logic Devices*). Consentendo all'utilizzatore la caratteristica della programmabilità, la prima generazione di questi chip poteva rimpiazzare fino a una diecina di integrati logici discreti.
Con il miglioramento dell'integrazione, le PLD furono in grado di sostituire sempre più numerose funzioni logiche standard. Oggi, le più avanzate CPLD (*Complex PLD*) contengono decine di migliaia di gates e la loro capacità equivalente corrisponde a quella di centinaia di dispositivi della serie 74.
Tuttavia, le prime generazioni di CPLD costavano ancora molto. Se consideriamo il processo di fabbricazione, in sintesi il prezzo del dispositivo finale è determinato da quanto si riesce a partizionare la fetta di silicio da cui si parte (figura 5). Se, ad esempio, il wafer costa 3000 dollari e si ottengono 300 matrici (Die) per wafer, il costo di ciascun dispositivo si aggira sui 10 $, a cui va sommato il contributo del package e delle successive lavorazioni.

La resa del processo produttivo

Il processo di produzione di un circuito integrato segue regole molto precise e tutti i parametri vengono tenuti rigidamente sotto controllo per ridurre al minimo le imperfezioni. Si può dire che la massima dimensione di un circuito integrato, secondo la regola empirica di Rent,

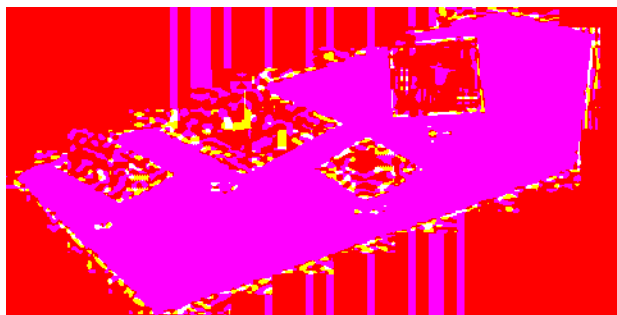



Figura 6 Evoluzione dei packages

dipende dalla resa ottenibile nel processo di fabbricazione (tipicamente dell' 80%) e dal massimo numero di pin di I/O realizzabili.

Di pari passo con l'aumento della complessità, anche il package dei circuiti integrati ha subíto una progressiva evoluzione (figura 6), volta ad accrescere sempre più il numero di pin di I/O a disposizione. Dal package Dual-In-Line iniziale, si è giunti al moderno Ball-Grid-Array, per il cui montaggio sono necessarie specifiche attrezzature.

L'introduzione di nuovi materiali e di tecniche innovative di fabbricazione ha consentito un progressivo aumento della densità dei dispositivi (figura 7).
Le densità di integrazione ottenute con le strutture regolari, quali memorie e logiche programmabili, hanno consentito di incrementare ulteriormente le prestazioni raggiunte nei circuiti logici complessi, come i microprocessori.

In particolare, le tecnologie basate sul MOSFET (CMOS) attualmente sono preponderanti nel mercato dei semiconduttori e l'utilizzo delle tecnologie bipolari è limitato a particolari applicazioni.

Costi

Tipicamente, per chip di grandi dimensioni i costi sono elevati a causa del fatto che i wafer di silicio possono avere delle difettosità e la probabilità che un difetto si verifichi entro un chip è tanto più alta quanto più è estesa l'area che lo stesso chip occupa sul wafer. Così, oltre un certo limite, il costo di un chip di grandi dimensioni

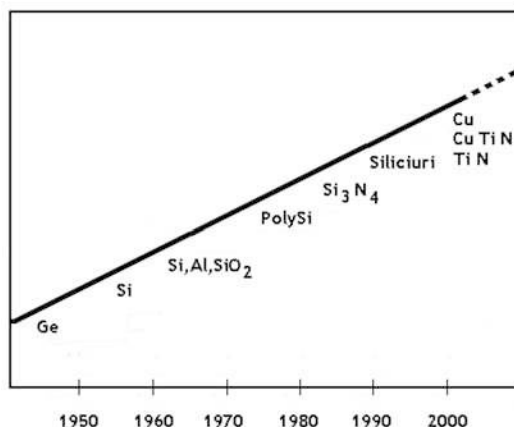



Figura 7 *Aumento della densità dei dispositivi in rapporto ai substrati utilizzati*

aumenta in maniera esponenziale (figura 8). Questi fattori, nel passato, hanno condizionato gli alti costi produttivi delle CPLD rispetto ai dispositivi discreti TTL, che occupavano un'area molto inferiore sul silicio.

Inoltre, come già accennato, in aggiunta al costo del substrato del chip, vi sono altri contributi da considerare, tra cui il package e il test, che concorrono a determinare il costo finale. La figura 9 mostra un diagramma che fotografa la situazione nel 1985.

Tradizionalmente, gli alti costi hanno ristretto l'impiego delle CPLD soprattutto ai prototipi di pre-produzione o alle applicazioni in volumi limitati. Oggi la situazione è cambiata radicalmente e i campi di applicazione sono sempre più estesi.

Quando la serie 74 fu introdotta, il prezzo di partenza medio per un singolo dispositivo era di circa 1000 $ (1963). Con il procedere della tecnologia nel corso degli anni, il prezzo scese a circa 25 $ (1968).
Allo stato attuale, i progressi nei processi di fabbricazione dei semiconduttori e nelle tecniche di packaging hanno fatto sì che oggi, per un circuito con oltre circa 8 integrati discreti TTL, sia più economicamente conveniente utilizzare una CPLD (figura 10) rispetto all'equivalente set di chip.

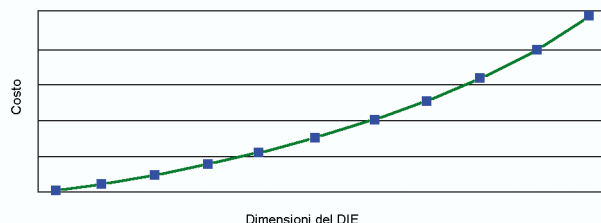



Figura 8 *Andamento del costo in rapporto alle dimensioni*

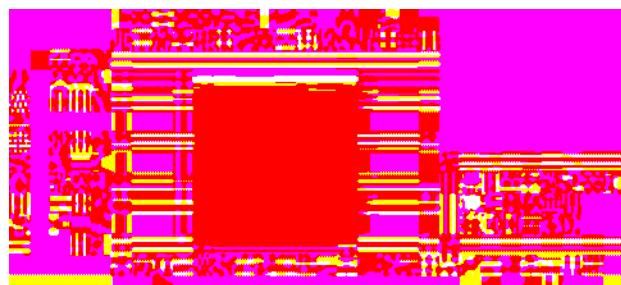



Figura 9 *Costo totale dell'IC*

Altri fattori

Fino ad ora abbiamo considerato il solo fattore costo ma, nel progetto di un circuito, vi sono altri elementi da tenere in considerazione, come il risparmio di area occupata sul pcb, la riduzione nel numero di piste di interconnessione, la riprogrammabilità, l'affidabilità, il ridotto time-to-market; tutti fattori che vedono protagoniste le moderne CPLD.

In sintesi

Dalla loro introduzione nei tardi anni '70, i dispositivi logici programmabili si sono dimostrati estremamente competitivi. Oggi rappresentano uno dei settori a crescita più dinamica dell'industria dei semiconduttori.

FLUSSO CAD PER PROGRAMMABILI SPLD

Quando ci si accinge a progettare un circuito che fa uso di componenti programmabili, è essenziale disporre di un sistema di sviluppo software che supporti i componenti utiliizzati. I tools CAD sono importanti non solo per i dispositivi complessi, come le CPLD o le FPGA, ma anche per i più semplici SPLD (Simple PLD).
Un tipico sistema CAD per SPLD consiste in un ambiente che include gli strumenti software per effettuare i seguenti passi: l'immissione o *entry* del disegno, *l'ottimizzazione* logica, il *fitting* entro il dispositivo, la *simulazione* e infine la *configurazione*.

Il flusso di progetto è illustrato in figura 11 e indica i passaggi da uno step all'altro.
La entry del progetto può essere fatta per mezzo di un applicativo grafico di schematica o mediante una descrizione testuale in un linguaggio di descrizione dell' hardware (Hardware Description Language, HDL). Il linguaggio può essere ABEL, AHDL, VHDL, VERILOG o altro; in commercio esistono diversi tool che consentono di progettare in HDL.

Dal momento che la entry logica è in forma di sorgente (quindi non ottimizzata), si impiegano poi degli algoritmi opportuni per ottimizzare il circuito; dopodichè, altri algoritmi analizzano le risultanti equazioni logiche e le "fittano", cioè le adattano al meglio, entro la SPLD.

Successivamente viene effettuata la fase di simulazione, per verificare che le temporizzazioni siano
effettivamente quelle desiderate. Con la simulazione si visualizzano le uscite interessate e il loro andamento nel tempo. Se vi sono, ad esempio, dei ritardi che possono provocare errori, allora si andrà a rivedere la descrizione in HDL e si ripeterà il processo.

La fase di simulazione è utile, per non dire essenziale, per verificare la correttezza del progetto prima ancora di averlo fisicamente realizzato. L'utente può da qui rientrare più volte alla fase di entry per eliminare gli eventuali errori e ottimizzare il funzionamento del circuito logico.

Quando infine il progetto si riesce a simulare correttamente, si può procedere al trasferimento verso *l'unità di programmazione*, dove si programma la SPLD.

I passi di progetto per realizzare circuiti logici entro le CPLD, come vedremo nel seguito, sono molto simili a quelli per le SPLD, ma gli strumenti software sono molto più sofisticati.

Il massimo della complessità si raggiunge con i dispositivi FPGA. Non ci occuperemo di FPGA in questo corso; possiamo citare soltanto il fatto che, per una FPGA, il processo di "fitting" è ancora più articolato: necessita di una fase di "*mappatura*" (*mapping*) per collocare i gates logici di base entro i blocchi logici reali della FPGA, di un "*piazzamento*" (*placement*), per scegliere quali specifici blocchi vanno usati della

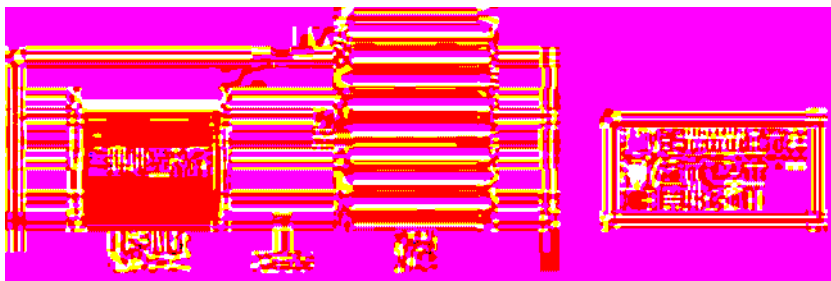



Figura 10 *Confronto tra il prezzo di un dispositivo 74 e quello di una CPLD*

FPGA e di un "*instradamento*" (*routing*) per distribuire i collegamenti interni che devono interconnettere i blocchi logici. Disponendo di queste complessità aggiuntive, i tool CAD possono richiedere tempi anche piuttosto lunghi per completare le varie fasi. Xilinx ha sviluppato il proprio "motore" ISE (Integrated-Software-Environment) il quale incorpora, in un unico ambiente di sviluppo, tutti i vari processi e permette di gestire le diverse tipologie di componenti: dalle CPLD delle serie XC9500 e Cool Runner alle FPGA delle serie Spartan e Virtex.

Figura 11 *Flusso di progetto per SPLD*

SPLD COMMERCIALI

Mattoni fondamentali per i progettisti hardware digitali da due decenni a questa parte, le Simple PLD costituiscono una categoria di dispositivi molto importante, apprezzate per avere prestazioni notevoli e bassi costi.
Agli albori dell'Elettronica moderna, i primi componenti programmabili erano memorie a sola lettura, come le PROM.
In seguito, nei primi anni 70, furono introdotte da Philips le **PLA** (Programmable Logic Array).
Questi dispositivi contengono un piano AND e un piano OR, entrambi programmabili.
Le PLA sono spesso integrate entro chip complessi, come i microprocessori. Non hanno avuto molto successo come dispositivi singoli perchè costose e dalle prestazioni limitate.

Le **PAL** (Programmable Array Logic), invece, sono risultate più semplici da realizzare, perchè il piano AND è programmabile ma il piano OR è fisso; sono meno costose rispetto alle PLA e, nel contempo, offrono migliori prestazioni, tanto che sono divenute molto popolari nelle applicazioni pratiche.
Tra le PAL più utilizzate, possiamo ricordare i dispositivi prodotti da AMD e conosciuti con le sigle 16R8 e 22V10 (figure 12a e 12b).

Entrambi sono da tempo standard industriali e sono prodotti da più fabbricanti con la stessa sigla.
Il nome "16R8" significa che la PAL ha al massimo 16 ingressi (sudddivisi in 8 ingressi dedicati e altri 8 configurabili come ingressi/uscite) e un massimo di 8 uscite. La "R" indica che ciascuna uscita può essere "Registrata", avendo un flip flop di tipo D. Similmente, la 22V10 ha un massimo di 22 ingressi e 10 uscite. Qui, la "V" sta a indicare che ciascuna uscita è "Versatile", "cioè può essere configurata in differenti modi, alcuni registrati e altri no (vedere figura 12c).

Un'altra SPLD molto nota è la classica EP610, di Altera. Questo dispositivo è simile, nella complessità, alle PAL ma offre una maggiore flessibilità, perchè le uscite vengono prodotte mediante

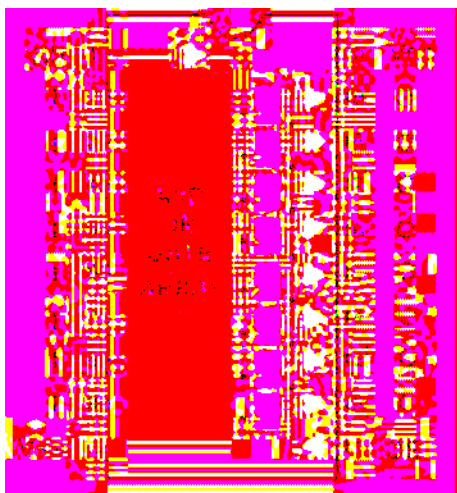



Figura 12a *Struttura interna della PAL16R8*

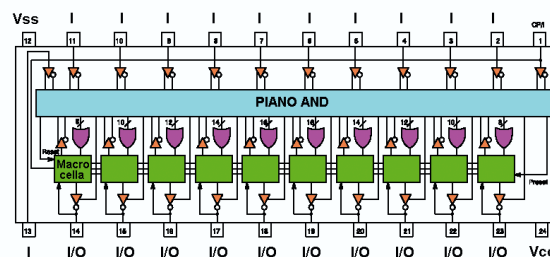



Figura 12b *Struttura interna della PAL22V10*

piani AND e OR di maggiore ampiezza. Nella EP610 le uscite possono essere registrate e i flip-flop sono configurabili di tipo D, T, JK o SR.

Oltre alle SPLD che abbiamo appena citato vi sono molti altri prodotti, messi a disposizione da diversi fabbricanti. Tutte le SPLD presentano caratteristiche simili, come il fatto di avere una struttura a "piani logici" (AND,OR,NOR o NAND), ma ciascun prodotto ha caratteristiche uniche che possono renderlo interessante per particolari applicazioni. Tra i vari produttori di SPLD ricordiamo AMD, Altera, Lattice, Cypress, Atmel, ecc..

CPLD COMMERCIALI

Prendiamo ora in considerazione alcuni tra i più noti fabbricanti di CPLD. Come abbiamo accennato in precedenza, la CPLD consiste di più blocchi di tipo *Simple PLD* riuniti su un singolo chip. Tuttavia, i dispositivi CPLD sono molto più sofisticati di quanto non siano le SPLD stesse, anche già a livello dei blocchi elementari simil-SPLD di cui sono composte. In questo paragrafo vedremo alcune delle più classiche CPLD commerciali e considereremo i campi di applicazione tipici di questi dispositivi. Teniamo presente che, data la rapida evoluzione in questo settore, per avere informazioni aggiornate è bene consultare i siti web dei vari produttori.

CPLD di Altera

Altera ha sviluppato tre famiglie che rientrano nella categoria delle CPLD: MAX 5000, MAX 7000 e MAX 9000.
Consideriamo in particolare la famiglia MAX 7000, poichè è ampiamente utilizzata ed offre capacità e prestazioni in velocità tipiche dello stato dell'arte attuale.
La serie MAX 5000 rappresenta la tecnologia più datata ed ha la maggiore convenienza economica, mentre la MAX 9000 è più simile alla MAX 7000 ed offre una capacità superiore in termini di gates equivalenti. L'architettura di Altera MAX 7000 è riportata in figura 13.

Essa contiene una matrice di blocchi, chiamati Logic Array Blocks (LAB) e piste di interconnessione denominate Programmable Interconnect Array (PIA). La struttura del PIA consente di collegare tra di loro le porte di ingresso e uscita dei LAB. Inoltre, permette di connettere gli ingressi e le uscite del chip direttamente ai LAB.
Un LAB può essere pensato come una struttura SPLD complessa, cosicchè l'intero chip si può considerare come una matrice di SPLD.

I dispositici MAX 7000 sono disponibili in tecnologia EPROM o EEPROM. Fino a poco tempo fa, i chip potevano essere programmati soltanto "out-of-circuit" e con un programmatore apposito. Dal '96 Altera ha prodotto la serie 7000 S, che è programmabile "in-circuit".

CPLD di Advanced Micro Devices (AMD)

Prima della cessione a Lattice della Divisione Logiche Programmabili, AMD metteva a disposizione una serie di CPLD comprendente 5 sottofamiglie denominate MACH 1, fino a MACH 5.
Ognuno di questi dispositivi contiene dei bloc-

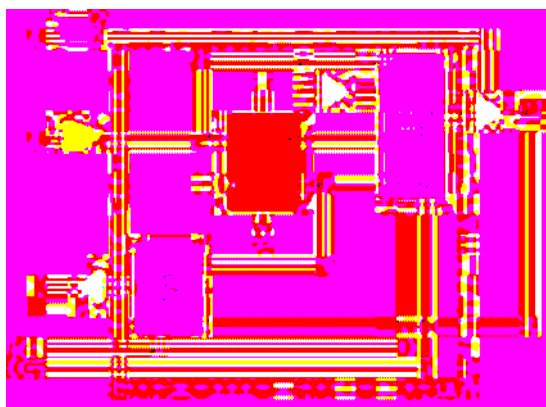



Figura 12c *Struttura della macrocella 22v10*

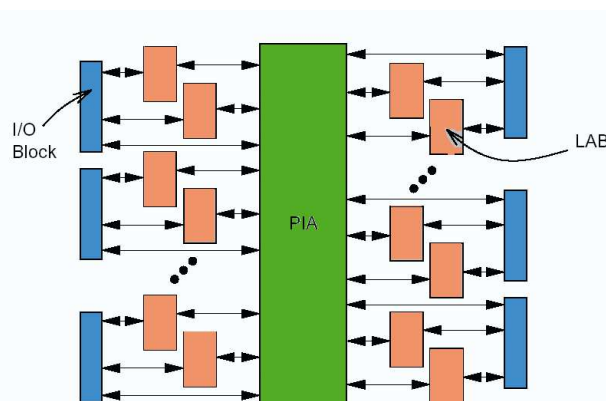



Figura 13 *Serie Altera MAX 7000*

chi *PAL-like*: i tipi MACH 1 e 2 contengono blocchi PAL del tipo 22V10, mentre MACH 3 e 4 dispongono di blocchi tipo 34V16. La serie MACH 5 è simile, ma offre prestazioni superiori in velocità.
Tutte le sotto-famiglie MACH si basano su tecnologia EEPROM e forniscono un ampio range di chip, dai più piccoli a basso costo, ai più potenti e costosi. Analizziamo in particolare la struttura della serie MACH 4. In figura 14 vediamo raffigurati i blocchi tipo PAL-34V16 e le interconnessioni che fanno capo alla matrice di connessione detta *Central Switch Matrix*.

Il chip ha una taglia che va da 6 a 16 blocchi *PAL-like*, che corrisponde a circa 2000 ÷ 5000 gates equivalenti ed è programmabile in-circuit.

CPLD di Lattice Semiconductor

Lattice, oltre a portare avanti le linee di prodotto già di AMD, ha sviluppato un proprio range completo di CPLD, con due linee principali: il tipo pLSI, che contiene alcune famiglie di tipo EEPROM, e il tipo ispLSI, che ha in più la programmabilità in-system. Le due tipologie si differenziano nella capacità e nella velocità.
La prima generazione di CPLD Lattice è rappresentata dalle serie 1000 pLSI e ispLSI. La struttura interna di questi chip consiste in una collezione di blocchi di tipo SPLD, descritti in seguito, e da una struttura di routing (detta *Global Routing Pool*) per collegare i blocchi tra di loro. La capacità va da circa 1200 a 4000 gates. Il ritardo da pin a pin è di 10 nanosecondi.
Lattice inoltre produce la serie 3000, che arriva a 5000 gates, con ritardi di circa 10-15 nsec. La funzionalità della serie 3000 è molto simile a quella della serie Mach 4 della AMD.
La serie 3000 supporta gli stili di progetto più moderni, come quello del *boundary* scan.
La struttura generale di un dispositivo pLSI o ispLSI è indicata in figura 15. Al contorno del chip si trovano le porte di I/O bidirezionali, che sono connesse sia ai *Generic Logic Blocks* (GLB) che al *Global Routing Pool* (GRP).
I GLB sono blocchi *PAL-like* che contengono un piano AND, l'area che interessa i termini di prodotto e le macrocelle. Il GRP contiene le piste che si svolgono attraverso l'intero chip e servono a collegare le linee di ingresso e uscita dei GLB tra loro.

CPLD della serie Cypress FLASH 370

Cypress ha sviluppato una famiglia di CPLD che è simile per certi aspetti sia ai dispositivi AMD sia a quelli Lattice. Le CPLD Cypress della serie FLASH 370 sono basate sulla tecnologia EEPROM di tipo FLASH e hanno prestazioni in velocità da 8.5 a 15 nsec da pin a pin.
I componenti di questa serie non sono programmabili in-system; in compenso, dispongono di più pin di I/O rispetto ai prodotti concorrenti. I chip più piccoli presentano 32 macrocelle e 32 pin di I/O, mentre i più grandi hanno 256 macrocelle e 256 I/O.
La figura 16 mostra come la FLASH 370 possieda la tipica architettura da CPLD, con diversi blocchi PAL-like e una matrice di interconnessione programmabile (PIM) tra di essi. In ogni blocco PAL-like vi è un piano AND che alimenta

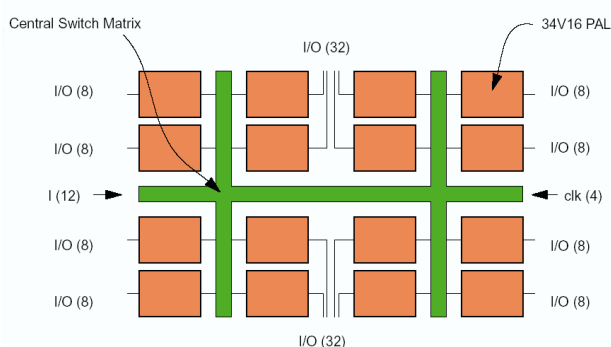



Figura 14 Struttura della CPLD AMD MACH 4

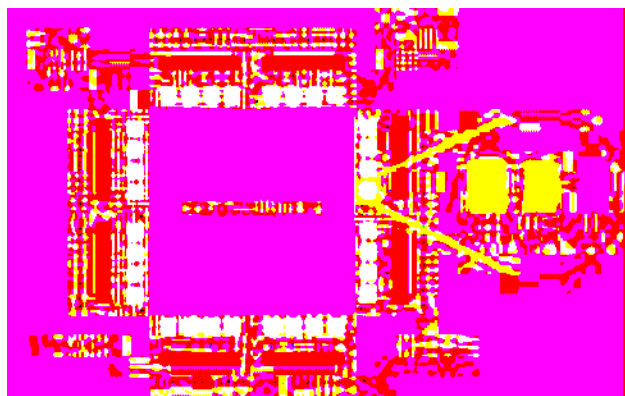



Figura 15 Struttura della CPLD Lattice pLSI

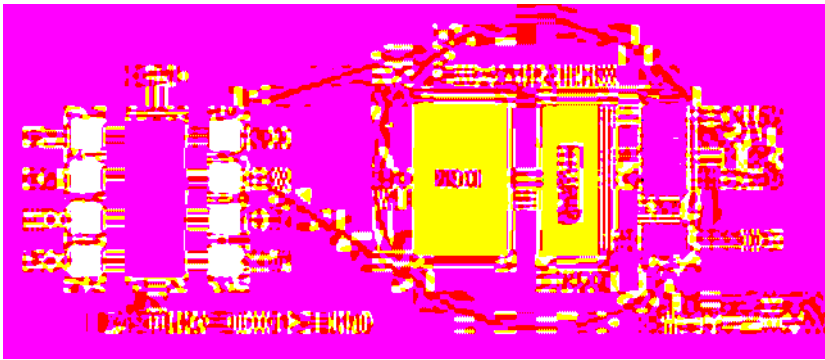



Figura 16 *Struttura della CPLD Cypress FLASH 370*

un *allocatore* di termini di prodotto, il quale pilota i gates di un piano logico di tipo OR.
Da notare che le macrocelle possono essere anche di tipo "sepolto" (*buried*). Questo comporta il non avere una connessione diretta verso i pin di I/O e ciò consente di poter utilizzare quei pin, che normalmente verrebbero pilotati dalle macrocelle, anche in funzione di ingressi. Questo tipo di flessibilità è presente in certe CPLD, ma non in tutte.

CPLD della serie Altera FLASH Logic

La serie FLASH logic di Altera, già nota come Intel FLEX logic, realizza la programmabilità in-system e mette a disposizione alcuni blocchi di Ram Statica sul chip, caratteristica questa unica tra i prodotti CPLD.
Nella figura 17, la parte a sinistra illustra l'architettura dei dispositivi della serie FLASH logic, caratterizzata da un set di blocchi di tipo PAL, chiamati *Configurable Function Blocks* (CFB); ciascuno di essi rappresenta una PAL 24V10 ottimizzata.

In termini di struttura-base, la FLASH logic è simile agli altri prodotti che abbiamo visto fino ad ora; tuttavia, la caratteristica peculiare di questi dispositivi consiste nel fatto che ciascun blocco PAL-like, anzichè essere assimilato a piani di tipo AND-OR, si può configurare come un blocco di Ram Statica da 10 nS di tempo di accesso.
Questo concetto è descritto nella parte destra della figura 17, in cui viene mostrato un blocco CFB configurato come una PAL e un'altro CFB configurato come Static Ram.

Figura 17 *Struttura della CPLD Altera FLASH logic*

Nella configurazione di tipo SRAM, il blocco PAL-like diventa una memoria in lettura/scrittura di 128 parole da 10 bit di ampiezza. Gli ingressi, che normalmente entrerebbero nel blocco *PAL-like*, in questo caso sono da intendersi come linee di indirizzi, dati e controlli per la Ram.
Notiamo che i flip-flop e i buffer tri-state sono ancora disponibili, anche se si utilizza la configurazione a Ram.

Il dispositivo FLASH logic deve il suo nome al fatto che i bit che costituiscono i piani logici AND-OR sono mappati su celle di EPROM o EEPROM.
Le celle di Ram statica vengono "caricate" con una copia di questa memoria non volatile (EPROM o EEPROM) ogni volta che il dispositivo viene acceso, ma sono le celle di Ram che controllano la configurazione del chip. È quindi possibile riconfigurare il chip, per mezzo della procedura *in-system*, mediante lo scaricamento di un nuovo file di configurazione che va a finire proprio nelle celle di Ram. Infine, il contenuto della Ram può essere ricopiato nella EEPROM, al fine di rendere non volatile la programmazione in-circuit.

CPLD della serie XC7000 di Xilinx Corp.

Sebbene Xilinx sia soprattutto un produttore di FPGA, tuttavia mette a disposizione un ampio spettro di componenti CPLD, che interessano una nicchia di mercato piuttosto ampia. Tra questi, possiamo menzionare la serie XC7000 e la più recente XC9500.
Della serie XC7000 vi sono due famiglie principali: la 7200 (in origine marchiata da Plus Logic

come Hiper CPLD) e la 7300, sviluppata in proprio da Xilinx. La prima comprende dispositivi di taglia piuttosto piccola, da circa 600 a 1500 gates di capacità, con ritardi di 25 nS pin-to-pin. I chip di entrambe le famiglie sono composti da una collezione di blocchi *SPLD-like*, con 9 macrocelle ciascuno. Le macrocelle della famiglia 7200 sono differenti da quelle tradizionali, in quanto ciascuna macrocella contiene due gate di tipo OR, che alimentano *un'Unità Aritmetico Logica* (ALU).
La ALU ha il pregio di poter generare una qualunque funzione dei suoi due ingressi; la sua uscita comanda un flip-flop configurabile.
La famiglia 7300 è una versione migliorativa della 7200 e fornisce una capacità superiore (fino a 3000 gates), oltre a una più elevata velocità.
La recente serie XC9500 offre prestazioni ulteriori: programmabilità in-system, ritardo garantito di 5 nS pin-to-pin e capacità fino a 6400 gates logici.

APPLICAZIONI DELLE CPLD

Diamo ora un'occhiata ai campi di applicazione tipici cui si adattano bene le architetture delle CPLD. Grazie all'elevata velocità e all'ampio spettro di prestazioni, le CPLD sono utilizzabili in un vasto assortimento di applicazioni, dalla semplice *Glue Logic* (termine usato per indicare la logica discreta che fa da "collante" al resto del circuito), alla prototipazione di piccoli gate array. Uno degli impieghi industriali più comuni, che rappresenta anche una forte motivazione alla crescita di questo mercato, è la conversione di progetti che contengono gates obsoleti di standard logic, oppure parecchie SPLD, verso soluzioni composte da un piccolo numero di CPLD (figura 18).
Le CPLD possono realizzare funzioni piuttosto complesse, come ad esempio controllori grafici, controllori LAN, dispositivi UART, gestori di memoria CASH e molti altri. Come regola generale, i circuiti che fanno largo uso di gates AND,OR, ecc.. e non necessitano di un gran numero di flip-flop, sono buoni candidati per essere convertiti entro CPLD.
Uno dei vantaggi significativi delle CPLD è che esse forniscono una modalità semplice e veloce per la modifica del progetto, grazie alla riprogrammabilità. Con le CPLD programmabili in-system è anche possibile riconfigurare l'hardware senza dover spegnere il circuito. Un esempio può essere cambiare il protocollo in un circuito di comunicazione durante il suo funzionamento. Spesso, un progetto si può compattare con facilità entro i blocchi *SPLD-like* di una CPLD. Il risultato fornisce prestazioni in velocità più predicibili di quanto non si abbia nel caso di un progetto suddiviso in tante piccole porzioni circuitali. La predicibilità della implementazione di un circuito è uno dei principali vantaggi dell'architettura delle CPLD.

CRITERI DI SCELTA

Innanzi tutto, per scegliere la CPLD giusta per le nostre esigenze è utile porsi prima alcune domande sulle caratteristiche di cui abbiamo bisogno, così da individuare la famiglia di prodotto più adatta per la nostra applicazione. I criteri di scelta possono essere i seguenti:

Densità: ogni famiglia di CPLD possiede una propria capacità, espressa in termini di gates equivalenti, che dà una stima della densità del chip.

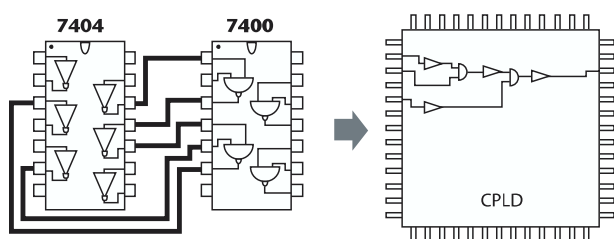



Figura 18 Equivalenza 74-cpld

Figura 19 Componenti della serie Xilinx XC9500

Numero di flip-flop: si fa una stima del numero di flip-flop di cui si ha bisogno per il progetto: contatori, macchine a stati, registri, latch. Il numero di macrocelle del dispositivo dovrà essere tale da coprire questo fabbisogno.

Numero di pin di I/O: quanti pin di ingresso e uscita ci servono?

Requisiti di velocità: qual'è il percorso combinatorio più veloce del nostro disegno? La risposta a questa domanda determina il tempo di propagazione (Tpd), in nanosecondi, del dispositivo da scegliere. Qual'è il più veloce circuito sequenziale richiesto? La risposta ci indica quale frequenza massima (Fmax) scegliere.

Package: abbiamo dei requisiti di carattere elettro-meccanico da soddisfare? Abbiamo bisogno del più piccolo package (Ball Grid Array) o possiamo utilizzare un più comune QFP? Stiamo forse realizzando un prototipo e quindi ci serve una CPLD su zoccolo, come per il PLCC?

Consumi: la nostra applicazione andrà alimentata a batteria o a celle solari? Abbiamo bisogno di una CPLD che consumi il meno possibile? Vi sono problemi di dissipazione termica?

Funzionalità a livello di sistema: il nostro circuito contiene componenti che sono alimentati a tensioni differenti? Dobbiamo interfacciare tra loro dispositivi con soglie diverse? Abbiamo dei clock critici da bufferare? Dobbiamo interfacciare la CPLD con microprocessori o memorie? Se sì, quali sono, per le memorie, i tempi di accesso? Ecc.

PANORAMICA INTRODUTTIVA SULLA SERIE XC9500

Attualmente Xilinx offre due categorie principali di prodotti CPLD, identificate nelle serie XC9500 e CoolRunner.
Le famiglie di CPLD XIlinx XC9500 sono intese per impieghi ad alte prestazioni e basso costo, in sistemi che richiedono sviluppi rapidi di progetto, lunga vita e robustezza intrinseca.
La serie XC9500 presenta densità che vanno da 36 a 288 macrocelle ed è disponibile nelle versioni a 2.5 Volt (XC9500XV), 3.3 Volt (XC9500XL) e 5 Volt (XC9500).
Questi dispositivi supportano la programmazione *in-system*, che permette di effettuare innumerevoli cicli di modifica durante le fasi di prototipazione. È possibile quindi effettuare decine di migliaia di cicli di programmazione/cancellazione. Le prestazioni in velocità sono elevate e il ritardo di propagazione è contenuto a 5 ns.
Si possono considerare di complemento alle FPGA Xilinx, "sorelle maggiori" ad alta densità, tanto che l'ambiente di sviluppo sofware è il medesimo. Il tool è già da alcuni anni scaricabile gratuitamente dal web e si chiama ISE Webpack. Il sito, all'indirizzo www.xilinx.com, è veramente una miniera di informazioni, sia per quanto riguarda i data sheet dei componenti, sia per gli strumenti software di sviluppo, per i quali esiste una ricca collezione di tutorials e di help on-line. Esistono perfino delle video-conferenze, tenute da esperti su temi specifici.

Reperibilità

Le CPLD della serie XC9500 (figura 19) sono rintracciabili, ad esempio, sul catalogo RS, all'indirizzo http://www.rs-components.it/.

Per identificare il part number del dispositivo possiamo fare riferimento a questo schema:

XC xx yy kk – zz YYY t

Dove **XC xx** identifica la serie Xilinx 9500, **yy** indica la capacità (36,72,..,288 macrocelle), **kk** indica il processo tecnologico (niente se +5V, XL se 3,3V, XV se 2,5V), **zz** è lo speed-grade in nanosecondi, **YYY** rappresenta il package e il numero di pin (PC=PLCC, PQ=PQFP, ...) e infine **t** indica il range di temperatura operativa (C=commerciale, I=industriale).

La famiglia a 5V XC9500

La famiglia di CPLD XC9500 contiene sei dispositivi, da 36 a 288 macrocelle di capacità, in un'ampia varietà di package. La tabella 1 ci mostra una panoramica di questi componenti.
I pin di I/O si possono interfacciare a sistemi sia a 3.3 che a 5 Volt e arrivano a supportare fino a 192 segnali.

Architettura flessibile del pin lock

Nel caso di modifiche l'architettura del chip,

insieme al software, è tale da consentire di conservare l'assegnazione iniziale del pinout, fissata nelle fasi iniziali di progetto, anche in presenza di consistenti cambiamenti nel codice.

Supporto JTAG IEEE 1149.1

La famiglia XC9500 è supportata da parecchi tool di sviluppo e di debug, che servono a sviluppare vettori di test per istruzioni del Boundary Scan allo scopo di analizzare, testare e debuggare gli eventuali malfunzionamenti del sistema.

La famiglia a 3.3V XC9500XL

I componenti della famiglia di CPLD XC9500XL sono indirizzati a sistemi avanzati che richiedono sviluppi rapidi di progetto, lunga durata e necessità di upgrade sul campo. Questa famiglia di dispositivi ISP (dotati cioè di In System Programming) fornisce ottime prestazioni e costi tra i più bassi nel settore. Le caratteristiche principali di questa famiglia sono le seguenti: bassi costi per macrocella, architettura a pin-lock avanzata, ritardo di 5 nS, frequenza massima di 178 Mhz, tre ingressi di

	36 macrocelle			72 macrocelle			108 macrocelle	144 macrocelle			216 macrocelle	288 macrocelle		
Caratteristiche	XC9536	XC9536XL	XC9536XV	XC9572	XC9572XL	XC9572XV	XC95108	XC95144	XC95144XL	XC95144XV	XC95216	XC95288	XC95288XL	XC95288XV
Speed Grades	5, 6, 10, 15	5, 7, 10	5, 7	7, 10, 15	5, 7, 10	5, 7	7, 10, 15, 20	7, 10, 15	5, 7, 10	5, 7	10, 15, 20	10, 15, 20	6, 7, 10	6, 7, 10
Max I/O	34	36	36	72	72	72	108	133	117	117	166	192	192	192
Vcc (Volts)	5	3.3	2.5	5	3.3	2.5	5	5	3.3	2.5	5	5	3.3	2.5
Package type														
VQ44	34	34	34		34	34								
PC44	34	34	34	34	34	34								
CS48	34	36	36		38	38								
VQ64		36			52									
PC84				69			69							
TQ100				72	72	72	81	81	81	81				
PQ100				72			81	81						
CS144									117	117			117	117
TQ144									117	117				
PQ160							108	133			133			
HQ208											166	168		
PQ208													168	168
BG256													192	
FG256													192	192
CS280													192	192
BG352											166	192		

***Tabella 1** Riassunto delle caratteristiche della serie XC9500*

clock globali, ridotti tempi di programmazione e cancellazione, eccetera.

La famiglia a 2,5V XC9500XV

La caratteristica principale di questa famiglia è rappresentata dall'alimentazione del core a 2,5 V, che pertanto offre consumi ridotti del 30% rispetto alla famiglia XL e costi inferiori.
Ne è prevista una futura versione a 275 Mhz.

LA SERIE COOL RUNNER

Le famiglie di CPLD Xilinx della serie CoolRunner sono particolarmente interessanti. Caratterizzate dai consumi estremamente ridotti, impiegano la tecnologia *Fast Zero Power™*, che porta ad assorbire meno di 100 uA in standby mode. In una dimostrazione ho visto alimentare una di queste CPLD con una batteria realizzata con un paio di agrumi (pompelmi) in cui erano state inserite placche di rame e zinco. Sono adatte per impieghi in apparecchi portatili a batteria, come telefoni cellulari, giochi elettronici, PC portatili, eccetera.

La serie contiene due dispositivi: la XPLA3 (con core a 3,3 V) e la Cool-Runner-II, alimentata a 1,8 V. L'architettura della XPLA3 consente di potersi interfacciare con dispositivi a 5V.
La tabella 2 riassume le caratteristiche di questi componenti.

ALCUNE CONSIDERAZIONI PRATICHE

Per quanto riguarda le sperimentazioni hobbistiche, sono preferibili quelle CPLD che si possono montare su zoccolo e che si riescono ad ordinare anche in quantità limitate e a costi contenuti.
La nostra serie di riferimento sarà la XC9500, in package PLCC a 44 pin.
Come tool software per la sintesi prenderemo in considerazione ISE Webpack, la cui disponibilità è gratuita insieme ai periodici aggiornamenti.

PROSSIMAMENTE

Nella prossima puntata tratteremo in maniera approfondita le CPLD XC9500, l'ambiente di sviluppo ISE WebPack e alcuni esempi pratici.

	XCR 3032XL	XCR 3064XL	XCR 3128XL	XCR 3256XL	XCR 3384XL	XCR 3512XL
Macrocells	32	64	128	256	384	512
Usable Gates	750	1,500	3,000	6,000	9,000	12,000
Tpd(Ns)	4.5	5.5	5.5	7.0	7.0	7.0
ƒSYS (MHz)	213	192	175	154	135	135
ICCSB(↔A)	17	17	17	18	18	18
Package	User I/O	User I/O	User I/O	User I/O	User I/O	User I/O
VQ44	36	36				
PC44	36	36				
CS48	36	40				
CP56		48				
VQ100		68	84			
CS144			108			
TQ144			108	120	118	
PQ208				164	172	180
FT256				164	212	212
CS280				164		
FG324					220	260

Tabella 2 Serie Cool-Runner

L'amplificatore operazionale dalla A alla Z

La versatilità dell'op-amp trova una esaltazione nelle innumerevoli possibilità di impiego in alternata. E in entrambe le configurazioni non invertente e invertente.

7.1 GENERALITÀ

L'amplificatore operazionale quale componente base nelle applicazioni lineari, viene utilizzato sia in continua che in alternata. Nel primo caso, come si è visto negli articoli precedenti, la connessione fra la sorgente di segnale e l'ingresso dell'operazionale, così come fra l'uscita dell'operazionale e il carico, è diretta.

Nel secondo caso le stesse connessioni vengono realizzate tramite condensatori di opportuna capacità.

Questi condensatori fissano la frequenza di taglio inferiore f_L della banda passante B il cui limite superiore, ossia la frequenza di taglio f_H, è in genere fissato da un altro condensatore in parallelo alla resistenza R_f. Si ha comunque:

$$B = f_H - f_L \qquad [7.1]$$

Da questa si evince che un amplificatore è comunque un filtro passabanda che lascia passare, a –3 dB, i segnali la cui frequenza è compresa fra f_L e f_H.

La connessione tramite condensatori consente di modificare entro ampi limiti l'ampiezza della banda passante e, d'altra parte, elimina eventuali componenti continue che potrebbero essere presenti nel segnale-sorgente o, nel caso di stadi in cascata, componenti continue, come, per esempio la tensione di offset, presenti in uno o più stadi.

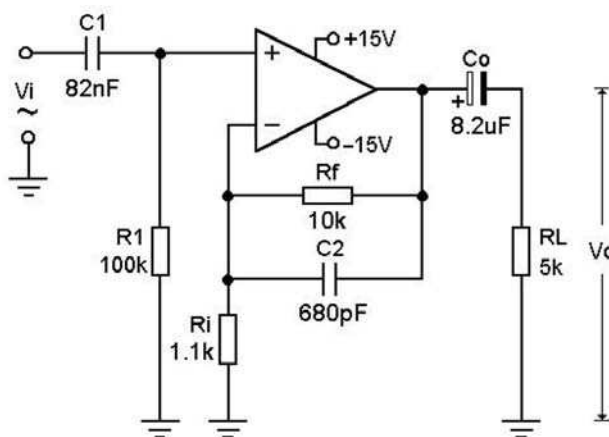



Figura 7.1 *Amplificatore con op-amp non invertente in c.a.*

7.2 AMPLIFICATORE CON OP-AMP NON INVERTENTE IN ALTERNATA

La figura 7.1 mostra una prima configurazione dell'op-amp non invertente in alternata. Il condensatore C_1, unitamente alla resistenza R_1, che oltre a fornire il ritorno a massa per la corrente di polarizzazione dell'input (+), fissa col suo valore la resistenza di ingresso, costituisce un filtro passa-alto con frequenza di taglio:

$$f_L = \frac{1}{2\pi C_1 R_1} = \frac{0,159}{C_1 R_1} \qquad [7.2]$$

Per il valore da attribuire alla capacità C_1 si avrà quindi:

$$C_1 \geq \frac{1}{2\pi f_L R_1} = \frac{0,159}{f_L R_1} \qquad [7.3]$$

Analogamente per il valore da attribuire alla capacità C_o si avrà:

$$C_0 \geq \frac{1}{2\pi f_L R_L} = \frac{0,159}{f_L R_L} \qquad [7.4]$$

In pratica, relativamente alla capacità C_o il valore teorico ricavato tramite la [7.4] lo si moltiplica almeno per cinque. Ciò equivale a porre la reattanza di C_o pari a 1/5 del valore della resistenza di carico R_L.

L'operazionale in corrente alternata

di Nico Grilloni
(n.grilloni@farelettronica.com)

Pertanto l'espressione [7.4] diviene:

$$C_0 = \frac{0,159}{f_L (R_L/5)} = \frac{0,796}{f_L R_L} \qquad [7.5]$$

Considerando, per esempio, la banda audio che si estende da 20 Hz a 20 kHz, per f_L = 20 Hz, R_1 = 100 kΩ e R_L = 5 kΩ, dalla [7.3] e [7.5] per le capacità C_1 e C_o si ha quindi:

$$C_1 \geq 0,159 / (20 \text{ X } 100000) = 79,5 \text{ nF}$$

$$C_o \geq 0,796 / (20 \text{ X } 5000) = 7,96 \text{ µF}$$

Normalizzando, si porrà C_1 = 82 nF e C_o = 8,2 µF. Le resistenze R_f e R_i del canale di reazione si calcolano come già si è visto in precedenza. Qui sono state calcolate per un guadagno A_{CL} = 10 e quindi posta la R_f = 10 kΩ, essendo $A_{CL} = 1 + (R_f / R_i)$, si è calcolata la R_i:

$$R_i = R_f / (A_{CL} - 1) = 10000 / 9 = 1,1 \text{ kΩ}$$

L'integrato utilizzato in simulazione è l'LF353 il cui stadio differenziale di ingresso è a Fet.
Ricorrendo a un op-amp con lo stadio di ingresso a BJT si sarebbe dovuta porre la R_1 pari al parallelo della R_f con la R_i (~ 1 kΩ) e ricalcolare quindi C_1 (più avanti si vedrà come sia possibile svincolare il valore della resistenza R_1, dai valori attribuiti alle resistenze R_f e R_i).
Il condensatore C_2 limita la banda superiormente e il suo valore va quindi calcolato in funzione della frequenza di taglio f_H. Essendo:

$$f_H = \frac{1}{2\pi \, C_2 \, R_f} = \frac{0,159}{C_2 R_f}$$

Per C_2 si userà l'espressione:

$$C_2 \leq \frac{0,159}{f_H \, R_f} \qquad [7.6]$$

Per R_f = 10 kΩ e f_H = 20 kHz (limite superiore della banda audio) si ricava:

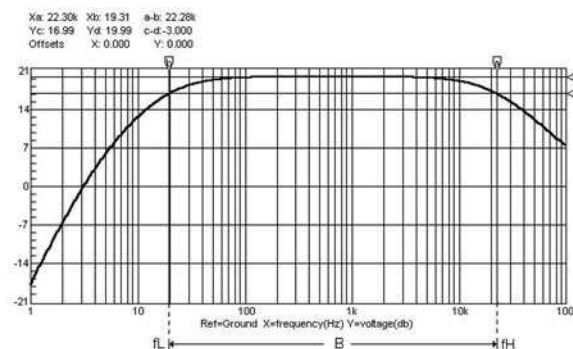



Figura 7.2 Risposta in frequenza dello stadio amplificatore di cui alla figura 7.1

$$C_2 = 0,159 / (20000 \text{ X } 10000) = 795 \text{ pF}$$

Il valore normalizzato inferiormente più prossimo è 680 pF. In assenza della capacità C_2, la frequenza di taglio superiore dipenderà soltanto dal guadagno assegnato allo stadio amplificatore (ossia dal valore attribuito alle resistenze R_f e R_i) e dal guadagno ad anello aperto dell'operazionale utilizzato. La figura 7.2 riporta la risposta in frequenza dello stadio di cui alla figura 7.1. La frequenza di taglio inferiore f_L è di circa 20 Hz (X_b = 19,31 Hz) e la frequenza di taglio superiore f_H è di circa 22 kHz (X_a = 22,3 kHz). La distanza fra il tratto orizzontale della curva di risposta e il livello 0 dB corrispondente al livello del segnale di ingresso, ossia l'ordinata Yd = 20 dB indica il guadagno dello stadio:

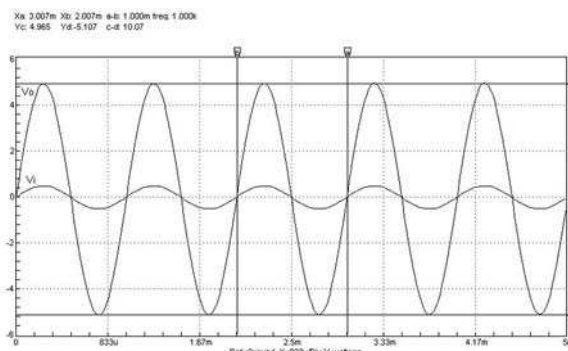



Figura 7.3 Risposta in ampiezza dello stadio amplificatore di cui alla figura 7.1

a 20 dB corrisponde un guadagno numericamente eguale a 10.
La distanza fra i marker verticali *a* e *b* indica l'ampiezza della banda passante *B* che, per la [7.1] vale:

$$B = 22300 - 19,31 = 22,28 \text{ kHz}$$

La figura 7.3 riporta quindi l'andamento delle tensioni V_i e V_o. Alla prima è stato attribuito un valore di 1 V da picco a picco.
L'amplificazione dello stadio porta la tensione V_o di uscita a 10 V da picco a picco indicati dalla posizione reciproca dei marker orizzontali *c* e *d*. È, infatti, $(c - d) = 10,07$ V.

7.2.1 Per rendere minimi I_{os} e V_{os} e le correnti di polarizzazione

L'amplificatore in alternata presenta il vantaggio di rendere ininfluenti, o quantomeno trascurabili, la corrente di offset e la tensione di offset. Come si è detto queste grandezze sono continue e pertanto i condensatori C_1 e C_o azzerano la possibilità, il primo, di risentire di eventuali componenti continue o dal generatore, o da uno stadio che preceda lo stadio amplificatore, e il secondo, di inviare componenti continue a uno stadio successivo. Non rimane che da "tagliare" l'eventuale componente continua di corrente che fluisce nella resistenza R_i. A questo fine è sufficiente disporre, come mostra la figura 7.4, un condensatore C_3 in serie alla resistenza R_i.

Figura 7.4 *Il condensatore C_3 in serie alla resistenza R_i consente l'azzeramento della tensione di offset*

Questa soluzione porta a valore unitario il guadagno in continua (l'amplificatore si comporta, rispetto alla continua, come inseguitore di tensione), mentre lascia inalterato il comportamento dello stadio in alternata. L'espressione utile per calcolare la capacità C_3 è:

$$C_3 = 0,159 / (f_L \cdot R_i) \quad [7.7]$$

Il valore così ricavato, che discende dall'eguaglianza fra la reattanza di C_3 e la resistenza R_i, si moltiplica almeno per cinque (o sei) in modo che la stessa reattanza sia trascurabile rispetto al valore della resistenza R_i. Con riferimento allo stadio della figura 7.4, per $f_L = 20$ Hz, si ha:

$$C_3 = 0,159 / (20 \text{ X } 1100) = 7,22 \text{ µF}$$

Moltiplicando per sei questo valore e normalizzando al valore superiore si può porre C_3 =47 µF.

7.2.2 La resistenza R_1 d'ingresso

Si è già affermato che, prevalentemente nel caso si utilizzino amplificatori operazionali con stadio di ingresso a BJT, è necessario attribuire alla resistenza R_1 un valore pari al parallelo delle resistenze R_f e R_i al fine di diminuire la tensione di offset dovuta alle correnti di polarizzazione.
Con riferimento, per esempio, all'amplificatore con guadagno pari a 10 di cui alla figura 7.1 nel quale è $R_1 = 100$ kΩ, poiché per R_1 vale l'espressione:

$$R_1 = (R_f \cdot R_i) / (R_f + R_i)$$

Si ha per R_f:

$$R_f = (R_i \cdot R_1) / (R_i - R_1)$$

Da questa si vede che dev'essere allora:

$$R_i > R_1$$

Poniamo quindi, per esempio, $R_i = 120$ kΩ.
Poiché si vuole un guadagno $A_{CL} = 10$ ed è $A_{CL} = 1 + (R_f / R_i)$ e quindi anche:

$$R_f = (A_{CL} - 1) R_i$$

Figura 7.5 La resistenza $R_2 = R_1$ rende pressoché eguale la resistenza verso massa dei due ingressi dell'op-amp

Sarebbe necessario attribuire alla resistenza R_f il valore:

$$R_f = 9 \times 120000 = 1\ M\Omega$$

Ma come si è già chiarito, è opportuno contenere il valore della resistenza R_f dal momento che la tensione di offset è ad essa direttamente proporzionale.
D'altra parte, ponendo, per esempio, R_f = 10 kΩe R_i = 1,1 kΩ per avere sempre un guadagno pari a 10, dovendo essere la resistenza R_1 eguale al parallelo della R_f con la R_i, si avrebbe R_1 = 990 Ω. Questo valore fornisce una resistenza di ingresso piuttosto bassa ($R_{iCL} = R_1$) e del tutto inaccettabile nel caso in cui la sorgente di segnale dovesse avere una resistenza interna non trascurabile.
Da qui la necessità di svincolare la R_1, che fissa il valore della resistenza di ingresso dello stadio, dal valore delle resistenze R_f e R_i.
La soluzione è esposta nella figura 7.5 dove una resistenza $R_2 = R_1$ è posta fra l'ingresso invertente e il punto di congiunzione della R_f con la R_i. In tal modo entrambi gli ingressi vedono verso massa una resistenza pressoché eguale. L'ingresso (+) vede la R_1, mentre l'ingresso (–) vede la resistenza [R_2 + (R_i//R_f)] il cui valore è prossimo al valore della R_1 (~ 100,99 kΩ).
Pertanto, adottando la configurazione circuitale di cui alla figura 7.5, si può attribuire alla resistenza R_1 un valore qualsiasi anche molto elevato e, come si è detto, svincolato dai valori attribuiti alle resistenze R_f e R_i. Questa possibilità si rivela assai spesso utile in particolare in presenza di una sorgente di segnale di elevata impedenza interna.

7.3 AMPLIFICATORE CON OP-AMP INVERTENTE IN ALTERNATA

La figura 7.6 riporta la configurazione dell'amplificatore con op-amp in configurazione invertente in alternata. La resistenza di ingresso, come si è visto, è data dal valore attribuito alla resistenza R_i in funzione della quale si calcolerà la capacità del condensatore C_1. Si ha quindi l'espressione:

$$C_1 \geq 0{,}159 / (f_L \cdot R_i) \qquad [7.8]$$

dove f_L è la frequenza di taglio inferiore. Pertanto, volendo, per esempio, una resistenza di ingresso di 10 kΩ, si porrà R_i = 10 kΩ e tramite la [7.8] si calcolerà il valore della capacità C_1. Per esempio, per f_L = 20 Hz, si ha:

C1 ≥ 0,159 / (20 X 10000) = 0,79 μF
=> C_1 = 1 μF

Per il calcolo della capacità C_o valgono l'espressione [7.4] e le medesime considerazioni in merito all'incremento da dare al valore teorico calcolato. Si avrà quindi:

C_o ≥ 0,159 / (20 X 5000) = 1,59 μF
=> C_o = 3,9 μF (o 4,7 μF)

Ove sia opportuno limitare superiormente la banda passante ad una frequenza f_H prestabilita, si porrà in circuito il condensatore C_2 per la cui capacità si ricorrerà all'espressione [7.6]. Supponendo una f_H = 20 kHz si ha:

C_2 ≤ 0,159 / (20000 X 100000) = 79,5 pF
=> C_2 = 68 pF

La resistenza R_p si porrà, per quanto già detto, eguale al parallelo delle resistenze R_f e R_i (se R_f >>R_i, si può porre $R_p = R_i$ = 10 kΩ).
La figura 7.7 mostra la risposta in frequenza dell'amplificatore della figura 7.6. Le frequenze di taglio sono indicate dai marker verticali *a* (f_H = 21,54 kHz) e *b* (f_L =19,26 Hz). La distanza fra gli stessi marker indica la banda passante a –3 dB individuati dalla distanza (*c* – *d*) degli omonimi marker orizzontali.
L'ordinata Y_d = 20 dB, ossia la distanza fra il mar-

ker *d* e il livello 0 dB del segnale di ingresso, indica il guadagno A_{CL} dello stadio che, data la configurazione invertente, è dato, in modulo, dal rapporto (R_f /Ri). Attribuito il valore di 100 kΩ alla R_f si è calcolato il valore della R_i con l'espressione $R_i = R_f / A_{CL}$ ricavando quindi per un guadagno A_{CL} = 10 (20 dB):

$$R_i = 100000 / 10 = 10 \text{ k}\Omega$$

La figura 7.8 mostra le forme d'onda dei segnali in ingresso (V_i = 1 V da picco a picco con frequenza di 1 kHz) e di uscita (V_o = 10 V da picco a picco essendo il guadagno pari a 10). Queste sono simili alle forme d'onda dell'amplificatore non invertente (vedi figura 7.3). L'unica differenza è nella fase. In questo caso, infatti, data la configurazione invertente, la tensione V_o di uscita è in opposizione di fase rispetto alla tensione d'ingresso.

L'amplificazione è indicata dalla posizione dei marker orizzontali *c* e *d*. È infatti (*c* – *d*) = =10,01 V. La posizione reciproca dei marker verticali *a* e *b* indica la frequenza: è (*a* – *b*) = 1 kHz.

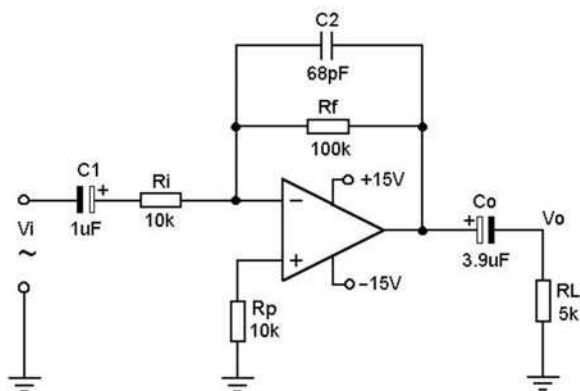



Figura 7.6 *Amplificatore con op-amp invertente in c. a.*

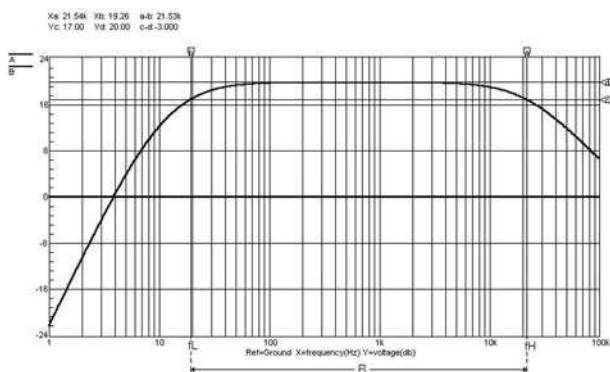



Figura 7.7 *Risposta in frequenza dello stadio amplificatore di cui alla figura 7.6*

7.3.1 La resistenza R_i di ingresso

Come si è già detto nel capitolo precedente, il valore della resistenza R_i di ingresso dev'essere sempre maggiore della resistenza interna R_s della sorgente (almeno 10 volte) in modo da non avere problemi di attenuazione nel trasferimento del segnale. La considerazione, infatti, che la resistenza di ingresso caratteristica dell'amplificatore operazionale è sempre molto elevata è irrilevante dal momento che l'effettiva resistenza di ingresso dello stadio amplificatore reazionato coincide col valore attribuito alla resistenza R_i.

Chiaramente si potrebbe sempre porre $R_i >> R_s$, ma nel caso di una sorgente di segnale che abbia una resistenza interna molto elevata, dell'ordine, per esempio di qualche centinaio di kΩ, ponendo $R_i > R_s$ si risolverebbe il problema di adattamento, ma, essendo il guadagno espresso dal rapporto R_f/R_i, si avrebbe un eccessivo valore per la R_f che, come si è detto, dev'essere sempre contenuto. Per esempio, se dovesse essere R_s = 200 kΩ, si dovrebbe porre per R_i, un valore di resistenza non inferiore a qualche MΩ, ma per un guadagno, per esempio, eguale a 10, si dovrebbe attribuire alla resistenza R_f un valore pari a 10 volte R_i. Il risultato sarebbe senz'altro inaccettabile.

Una soluzione efficace del problema è riportata nella figura 7.9. Lo stadio amplificatore, per esempio lo stadio della figura 7.6, è qui preceduto da un inseguitore di tensione che, quale intrinseca caratteristica, ha una resistenza di ingresso altissima ed è quindi in grado di evitare le perdite dovute a una resistenza elevata della sorgente.

Nell'amplificatore di cui alla figura 7.6, si è fatto

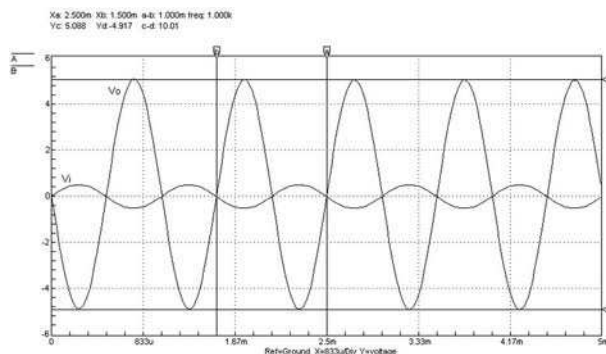



Figura 7.8 *Risposta in ampiezza dello stadio amplificatore di cui alla figura 7.6*

uso, in simulazione, dell'op-amp LF353 che, in simulazione, ha presentato una resistenza di ingresso R_{in} = 24 GΩ e un guadagno A_{OL} ad anello aperto pari a 10^5 (100 dB). Poiché la resistenza di ingresso dell'inseguitore di tensione è data dall'espressione:

$$R_{iCL} = R_{in} \cdot A_{OL}$$

per la resistenza di ingresso del voltage follower che, nella figura 7.9, precede lo stadio amplificatore, si ottiene:

$$R_{iCL} = 24 \text{ X } 109 \text{ X } 10^5 = 2{,}4 \text{ X } 10^{15} = 2{,}4 \text{ P}\Omega$$

(dove 1 PΩ = 1 Petaohm = 10^{15} Ω). Un'altra configurazione che consente elevata impedenza di ingresso e ampia possibilità di modificare il guadagno pur con valori contenuti della resistenza R_f, è riportata nella figura 7.10. Qui la rete a *T* posta fra l'uscita e l'ingresso invertente dell'operazionale è del tutto identica alla corrispondente della configurazione non invertente della figura 7.5. La resistenza R_f in questo caso, è fornita dall'espressione:

$$R_f = \frac{R_1 (R_1+2R_2)}{R_2} \qquad [7.9]$$

Poiché è comunque $A_{CL} = -(R_f / R_i)$, per il guadagno si ha:

$$A_{CL} = \frac{V_o}{V_i} = -\frac{1}{R_i} \cdot \frac{R_1 (R_1+2R_2)}{R_2} \qquad [7.10]$$

Con i valori di cui alla figura 7.10 si ha una R_f = 56,4 Ω. Il guadagno A_{CL}, essendo R_i =10 kΩ, vale dunque:

$$A_{CL} = R_f / R_i = 56400 / 10000 = 5{,}64$$

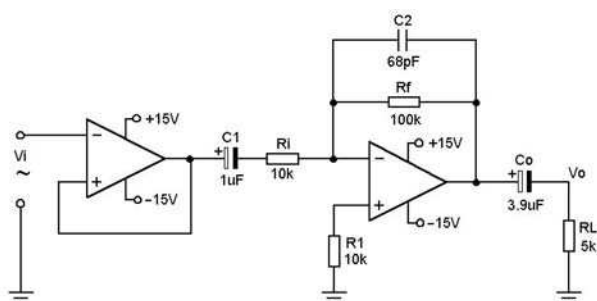



Figura 7.9 *L'inseguitore di tensione posto a monte dell'amplificatore rende l'impedenza di ingresso indipendente dalla impedenza interna della sorgente del segnale V_i*

La resistenza R_p di bilanciamento delle correnti di polarizzazione di ingresso deve essere eguale a $[(R_1//R_2) + R_1]//R_i$. Con i valori riportati in figura si ha R_p = 3389 Ω. Si può porre R_p =3,3 kΩ. La figura 7.11 riporta, infine la curva di risposta in frequenza. L'ordinata *Yd* = 15 dB indica il guadagno dell'amplificatore. A 15 dB corrisponde un guadagno pari a 5,62. Questo valore si era già ricavato per via analitica tramite la [7.10].

La configurazione della figura 7.10 è talvolta definita circuito moltiplicatore della resistenza di ingresso.

Esempio 7.1

Si dimensioni un amplificatore in configurazione invertente con guadagno pari a 50 e resistenza di ingresso R_{iCL} = 100 kΩ.

Soluzione

Poiché si richiede una resistenza di ingresso di 100 kΩ, è necessario porre R_i = 100 kΩ. Ricorrendo al circuito classico riportato nella figura 7.6 per A_{CL} = 50 e R_i = 100 kΩ, si dovreb-

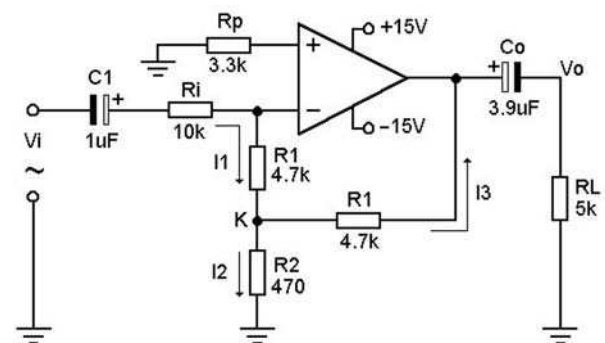



Figura 7.10 *La rete a T costituita dalle resistenze R_1 e della R_2 esalta l'impedenza di ingresso dello stadio*

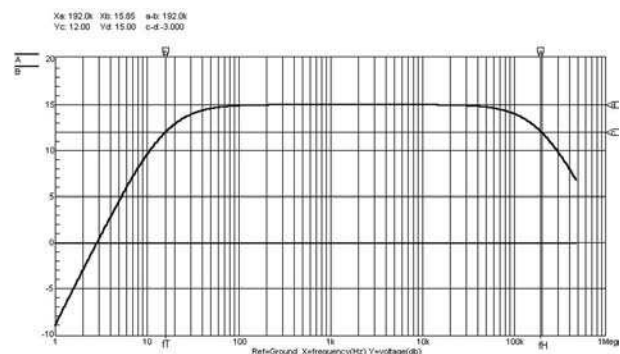



Figura 7.11 *Risposta in frequenza dello stadio amplificatore di cui alla figura 7.10*

be porre: $R_f = A_{CL}\ R_i = 50 \times 100000 = 5\ M\Omega$. Poiché questo valore della R_f è eccessivo, sarà opportuno o anteporre allo stadio un voltage follower o ricorrere allo schema circuitale della figura 7.10 ponendo comunque $R_i = 100\ k\Omega$.
Optando per quest'ultima soluzione, attribuito alle R_1 un valore compreso fra 10 kΩ e 250 kΩ, dalla [7.10] si ricava il valore da attribuire alla R_2. Si ha:

$$R_2 = \frac{R_1^2}{R_i A_{CL} - 2R_1}$$

Posto, per esempio, $R_1 = 47\ k\Omega$, si ricava per R_2:

$$R_2 = 47000^2 / [(100000 \times 50) - 94000] = 450{,}2\ \Omega$$

È infine necessario calcolare il valore da attribuire alla resistenza R_p di bilanciamento. Poiché l'ingresso invertente dell'op-amp "vede" una resistenza equivalente $[(R_1//R_2) + R_1]//R_i = 32{,}18\ k\Omega$, lo stesso valore, o un valore molto prossimo (33 kΩ), deve essere attribuito alla resistenza R_p.
La figura 7.12 riporta l'amplificatore qui dimensionato, mentre la figura 7.13 fornisce la relativa risposta in frequenza. In questa, l'ordinata *Yd* = 33,98 dB indica il guadagno. A 33,98 dB corrisponde un guadagno pari a 50.

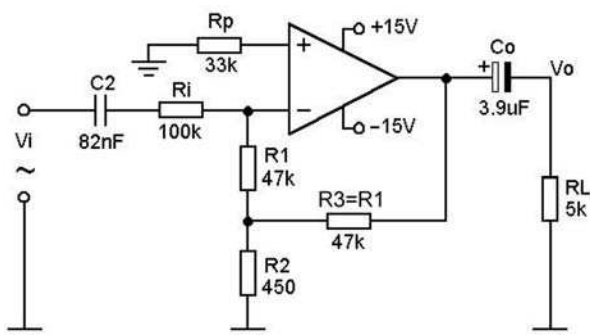



Figura 7.12 Stadio amplificatore dimensionato nell'esempio 7.1

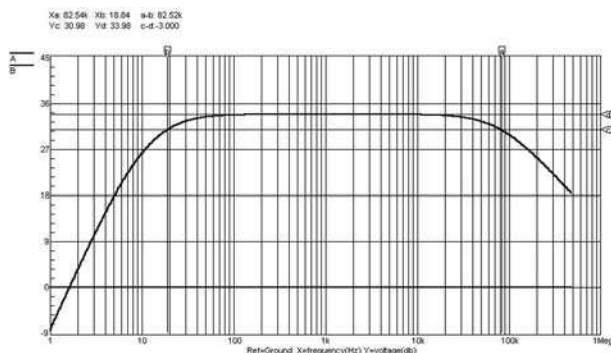



Figura 7.13 Risposta in frequenza dello stadio amplificatore di cui alla figura 7.12

7.4 IL GUADAGNO PER LE CONFIGURAZIONI AD ELEVATA R_{ICL}

7.4.1 Configurazione non invertente

Nell'amplificatore non invertente riportato nella figura 7.5 si è posta la resistenza $R_2 = R_1$ al fine di svincolare i valori delle resistenze R_i e R_f del canale di reazione dalla resistenza di ingresso dello stadio. Per questo amplificatore l'espressione del guadagno è comunque:

$$A_{CL} = 1 + (R_f / R_i) \qquad [7.11]$$

7.4.2 Configurazione invertente

Il guadagno per la configurazione invertente, posto $R_3 = R_1$, condizione non essenziale posta nell'amplificatore di cui alla figura 7.12, ha espressione:

$$A_{CL} = \frac{R_1^2 + 2R_1R_2}{R_iR_2} \qquad [7.12]$$

Mentre la resistenza equivalente R_f è, in linea del tutto generale, data dall'espressione:

$$R_f = \frac{R_1(R_2+R_3) + R_2R_3}{R_2} \qquad [7.13]$$

Nel caso si ponga $R_3 = R_1$, si avrà:

$$R_f = \frac{R_1^2 + 2R_1R_2}{R_2} \qquad [7.14]$$

Le configurazioni di cui alle figure 7.5 e 7.12, eliminati i condensatori, possono essere estese al regime in continua.

7.5 L'INSEGUITORE DI TENSIONE IN ALTERNATA

La necessità di porre in ingresso agli stadi fin qui analizzati un condensatore di disaccoppiamento di opportuna capacità, nasce dal presupposto che nel segnale applicato, oltre alla componente alternata, possa essere presente una componente continua. Il condensatore blocca questa componente e, se di opportuna capacità, lascia inalterato il segnale alternato.
Questo concetto che è basilare negli amplificatori in alternata, vale ovviamente anche per l'inseguitore di tensione.

Pertanto, con riferimento, per esempio, al voltage follower posto come buffer nel circuito di cui alla figura 7.9, ove sia presumibile che la sorgente di segnale, oltre ad avere una resistenza interna elevata, possa erogare anche una componente continua, sarà opportuno porre in serie all'ingresso non invertente un condensatore C_i e una resistenza R_1 così come illustra la figura 7.14.
La resistenza R_1, in presenza del condensatore, come si è visto negli stadi amplificatori non invertenti, è necessaria per fornire il ritorno a massa alla corrente di polarizzazione dell'input non invertente, ma, in pratica, col suo valore stabilisce la resistenza di ingresso dell'inseguitore la cui resistenza intrinseca è molto più elevata della stessa R_1 (per quanto grande questa possa essere).
Ciò significa che la presenza del ramo C_i-R_1, almeno in parte vanifica il fine per il quale il voltage follower era stato posto a monte dello stadio amplificatore che era di disporre di una resistenza di ingresso molto elevata.
Per ovviare a questo inconveniente si può ricorrere alla configurazione bootstrap, già molto frequente negli amplificatori a BJT e a Fet. Questa configurazione, riportata nella figura 7.15, consente di avere una resistenza di ingresso che è funzione del rapporto A_{OL}/A_{CL}.
Essendo per l'inseguitore di tensione, $A_{CL} = 1$, la resistenza di ingresso dello stadio sarà sempre notevolmente elevata. Il guadagno unitario implica poi che sia $V_o = V_i$ e poiché il segnale di uscita V_o viene riportato per via capacitiva all'estremo inferiore della R_1 al cui estremo superiore è applicato il segnale di ingresso V_i, nella stessa resistenza, essendo gli estremi equipotenziali, non circola corrente. Ciò porta a valori elevatissimi la resistenza di ingresso del voltage follower.
La figura 7.16 riporta la risposta in frequenza dell'inseguitore di tensione di cui alla figura 7.14. La banda passante a –3 dB si estende fino a circa 858 kHz (posizione del marker a che indica un'ascissa Xa = 857,7 kHz). Si noti altresì che la resistenza di ingresso è così elevata da fornire, con un elettrolitico da 5 µF in ingresso (C_i), un andamento piatto della curva già a partire dalla continua. Nel diagramma ricavato in simulazione, infatti, l'andamento piatto inizia visibilmente dalla frequenza f = 1 Hz.

Al Lettore attento facciamo notare che delle espressioni [7.11] ÷ [7.14], per motivi di semplicità, non si è fornito il calcolo analitico. L'Autore è comunque sempre a disposizione di coloro che desiderassero conoscere come si perviene alle relative espressioni.

PER IL LETTORE

I diagrammi qui riportati sono stati ricavati con il software di simulazione CircuitMaker della Microcode Engineering Inc. – Utah – USA, che utilizza SPICE *(Simulation Program with Integrated Circuit Emphasis)* realizzato dalla Microcode Engineering Inc. – Utah – USA. Sito Internet: microcode.com.

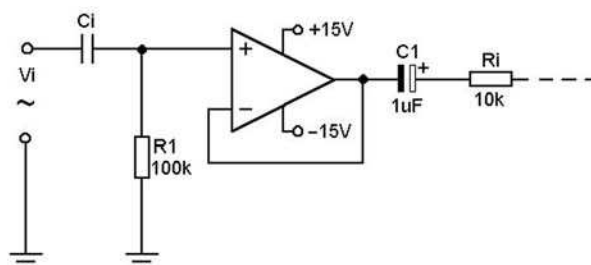



Figura 7.14 Voltage follower con op-amp in alternata

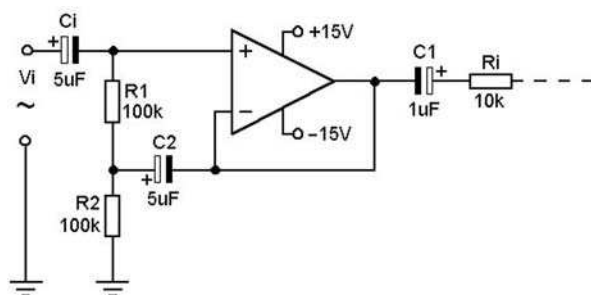



Figura 7.15 Voltage folloer con op-amp in alternata e con effetto bootstrap

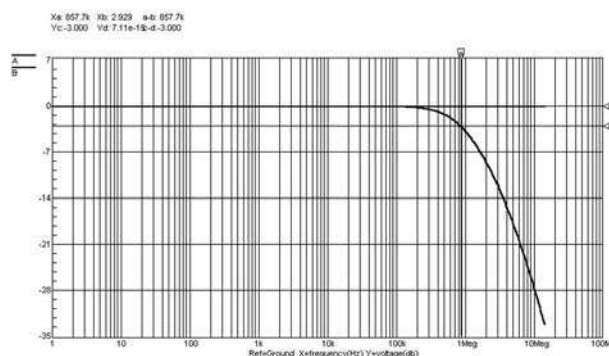



Figura 7.16 Risposta in frequenza del voltage follower di cui alla figura 7.15

Usare PROTEUS:

ISIS è il modulo di PROTEUS per la stesura degli schemi elettrici. In questo tutorial vedremo come utilizzarlo al meglio illustrando le funzioni principali. Ovviamente si presuppone che il pacchetto PROTEUS sia stato installato correttamente e che ISIS sia avviato dal menu Start di Windows. Il file progetto a cui fa riferimento questo articolo è ISISTUT.DSN che si trova nella cartella SAMPLES/TUTORIAL nel percorso di installazione di PROTEUS. La vostra avventura con PROTEUS inizia da qui...

UN RAPIDO SGUARDO ALL'EDITOR DI ISIS

L'ampia area di disegno nella finestra di ISIS è chiamata area di *Editing* - in tale area saranno posizionati e collegati i componenti o le parti. L'area più piccola in alto a destra è chiamata *Anteprima*. Normalmente l'*Anteprima* visualizza, come il nome stesso suggerisce, un'anteprima dell'intero disegno - il riquadro blu mostra i contorni del foglio di disegno, mentre il riquadro verde indica l'area del foglio visualizzato attualmente nell'area di *Editing*. Tuttavia, quando ISIS è nella modalità di selezione componente tramite *Object Selector*, l'*Anteprima* visualizza l'anteprima del componente stesso, come vedremo più avanti.
Nell'area di *Editing* è possibile visualizzare una griglia usando il comando *Griglia* nel menu *Visualizza*, oppure premendo "G", o ancora, cliccando sull'icona *Griglia* nella barra strumenti. La griglia è un utilissimo strumento che permette di allineare perfettamente i componenti e i vari collegamenti. Al di sotto della finestra *Anteprima* trovate l'*Object Selector* per mezzo del quale è possibile selezionare componenti, simboli e altri oggetti dalla libreria di parti.

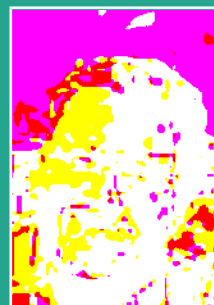
TECNICHE BASE PER IL DISEGNO DI UNO SCHEMA

Il circuito che andremo a disegnare è contenuto nel file ISISTUT.DSN nella cartella *samples* e pur apparendo complicato in prima battuta, esso è composto di alcune sezioni simili (le quattro sezioni filtro per la precisione) e ciò offrirà lo spunto per provare alcune funzioni dell'Editor per la copia di interi blocchi di circuito.
La prima cosa da fare è prelevare dalla libreria di parti i componenti che necessitano. Il procedimento è spiegato in dettaglio di seguito.

Prelevare i Componenti

Cliccate sul pulsante **P** in alto a sinistra del *Object Selector* come mostrato in figura 1. Ciò causa l'apertura della finestra di dialogo *Sfoglia Librerie.*
Potete anche aprire la suddetta finestra di dialogo tramite l'icona *sfoglia librerie* oppure mediante la sequenza di tasti appropriata (per default il tasto è "P", anche se questo può essere ridefinito tramite il comando *Imposta mappatura tastiera* nel menu *Sistema*). Il passo successivo è quello di trovare il componente desiderato nelle librerie.
La ricerca del componente può essere effettuata in

Figura 1 *Il pulsante Sfoglia Librerie*

Editor di ISIS: piazzamento e collegamento dei componenti

di Maurizio Del Corso
m.delcorso@farelettronica.com

molti modi. Nel caso il nome della parte stessa sia conosciuto è generalmente consigliabile iniziare da questo. Provate quindi ad inserire "741" nel campo di ricerca *Parole chiave*. Troverete naturalmente molte parti in cui è contenuta la stringa "741", sia nel nome che nella descrizione, tuttavia è possibile restringere il campo di ricerca selezionando la categoria "Operational Amplifiers", così come mostrato in figura 2:

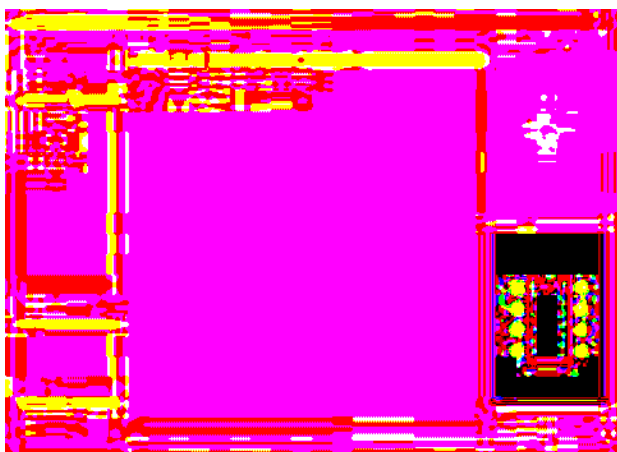



Figura 2 *Finestra di Dialogo Sfoglia librerie*

Finalmente, prelevate il componente "741" cliccando due volte sulla corrispondente riga della lista dei risultati. Il nome della parte apparirà nel *Object Selector*.
Abbiamo selezionato il "741" per il nostro circuito, ora abbiamo bisogno di alcuni resistori. Nella fattispecie, abbiamo necessità dei valori 1k, 10k, 12k, 15k, 56k, 68k e 100k. Ciò ci permette di familiarizzare con i vari meccanismi di ricerca accessibili mediante la finestra di dialogo *Sfoglia librerie*.
Una tecnica utilissima per trovare una parte in libreria è la ricerca mediante una chiave di ricerca coerente con la parte stessa. Provate a cambiare il testo della chiave in "12k resistor". Dovreste vedere una lista di risultati che contengono i termini della ricerca che avete effettuato. Per i nostri scopi il componente "MINRES12K" sarà più che sufficiente - selezionatelo dalla libreria allo stesso modo visto in precedenza.
Potremmo usare a questo punto lo stesso procedimento per i rimanenti resistori, tuttavia cercheremo di scoprire altre tecniche per la ricerca dei componenti disponibili. E' ragionevole pensare che la stessa logica e convenzione sia comune a tutte le parti in libreria. Infatti, se usiamo la chiave di ricerca "MINRES1" noi non facciamo altro che filtrare il risultato in modo tale da ottenere tutti i resistori MINRES che iniziano per 1 e quindi poter prelevare con facilità i valori da 1k, 10k, 15k e 100k.
Un altro metodo più generico è quello di cercare la parte desiderata attraverso il sistema di indici di categoria.
Ciò è utile quando elencando la lista delle parti disponibili non siete sicuri del nome della parte o della sua descrizione. Provate quindi a cancellare il testo dal campo *Parole chiave* e selezionate la Categoria "*Resistors*". Scorrendo i risultati verso il basso dovreste vedere la serie di resistori MINRES. Selezionate e prelevate i valori da 56k e 68k all'interno del vostro circuito nel modo ormai consueto ed infine chiudete la finestra di dialogo.
Val la pena di notare come si possa ricorrere a queste tecniche in tandem. Ad esempio, potreste aver introdotto "1k" nel campo di descrizione e poi aver selezionata la categoria *Resistors* per rifinire i risultati così ottenuti.
La descrizione fornita (per completezza) dei vari metodi di ricerca di un componente non deve spaventare; troverete velocemente il meccanismo a voi più consono ed il prelievo dei componenti dalla libreria per i vostri circuiti diventerà un problema di secondaria importanza.
Ora che abbiamo prelevato i nostri componen-

ti, sposteremo l'attenzione su come posizionarli nel nostro circuito.

Piazzamento dei Componenti nel Circuito

Abbiamo ora la necessità di posizionare i componenti prelevati all'interno dell'area di disegno - l'area di *Editing*. La sezione più semplice da cui iniziare è il circuito di buffer in alto a sinistra del circuito che stiamo considerando. Tale sezione è riprodotta in più dettaglio in figura 3. Selezionate con il mouse "741" nella "*Object Selector*". Dovreste notare che l'anteprima del componente è visualizzata nella finestra di *Anteprima*.
Ora spostate il puntatore del mouse al centro dell'area di *Editing*. Premete e tenete premuto il pulsante sinistro del mouse. Dovrebbe apparire sotto il puntatore il contorno di un op-amp che potete trascinare muovendo il mouse. Quando rilasciate il pulsante, la parte sarà posizionata e ridisegnata completamente. Posizionate l'op-amp al centro dell'area di *Editing*.
Selezionate ora la parte MINRES1K e piazzate un resistore esattamente sopra l'op-amp, come nella sezione ingrandita mostrata in figura 3.
Cliccate con il pulsante sinistro sull'icona *Ruota* senso anti-orario (figura 4); notate che l'anteprima nella finestra *Anteprima* mostra il resistore ruotato di 90°. Infine, piazzate allo stesso modo il secondo resistore (verticale), R2.

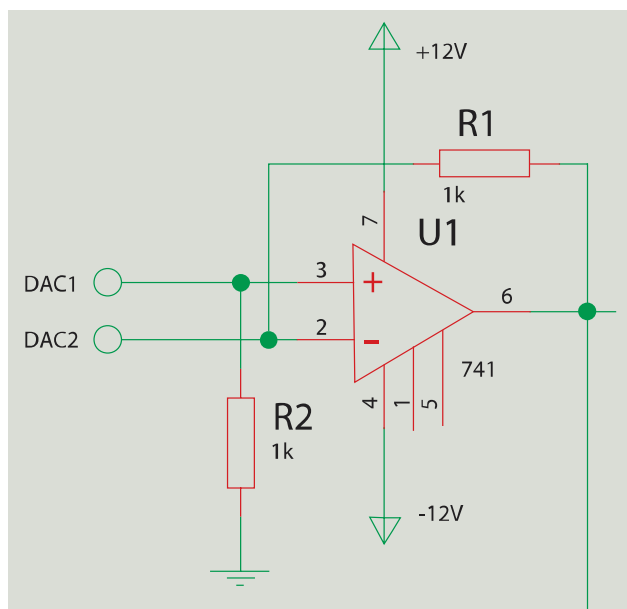



Figura 3 Una vista dettagliata della sezione iniziale del circuito da disegnare

In ISIS, gli oggetti già piazzati da editare successivamente sono evidenziati per questo scopo. Per comprenderne il significato, provate a puntare con il mouse l'op-amp e cliccate con il tasto destro del mouse. Ciò evidenzia l'oggetto, che viene disegnato in risalto rispetto a quelli non evidenziati.

Figura 4 L'icona Ruota con il senso di rotazione anti-orario selezionato

Ora, tenendo ancora il puntatore del mouse sull'oggetto, premete e tenete premuto il pulsante sinistro del mouse ed infine spostate l'oggetto dove desiderato. Questo è uno dei modi per spostare gli oggetti. Rilasciate il pulsante sinistro e premete sull'op-amp quello destro. Cliccando il pulsante destro su un oggetto evidenziato si avrà come effetto quello di cancellarlo. Selezionate *Annulla* dal menu *Modifica* (o premete "Ctrl-Z") per riottenere l'oggetto cancellato. Ora, evidenziate ancora una volta l'op-amp, cliccate sulle icone *Ruota*, sia per la rotazione oraria che anti-oraria e notate gli effetti sull'op-amp stesso. L'angolo di rotazione dell'ultimo oggetto evidenziato può essere cambiato anche in questo modo; l'icona *Specchia* può ugualmente essere utilizzata per ruotare l'ultimo oggetto evidenziato. Forti della pratica acquisita, dovreste essere in grado ora di spostare i tre componenti fin qui posizionati sino a trovarne il piazzamento ottimale così come da disegno. Una volta terminato il posizionamento delle parti evidenziate, spostate il puntatore del mouse in un area libera da componenti dell'area di *Editing* e cliccate il tasto destro del mouse. Ciò avrà l'effetto di riportare i componenti al loro stato normale non evidenziato.

Collegamento dei Componenti in un Circuito

Passiamo ora a collegare i componenti piazzati. ISIS ha un sistema intelligente per il cablaggio dei collegamenti chiamato *Wire*

Auto Router (WAR).
Ciò significa che non occorre, come in altri CAD, ricorrere ad una modalità di "wiring" - tutto ciò che dovete fare è puntare con il mouse su un pin da collegare e premere il pulsante sinistro. Provate a puntare al terminale in alto di R2 e cliccate il pulsante sinistro. ISIS comprende che state puntando ad un pin di un componente e quindi deduce che volete collegarlo. ISIS visualizza allora una retta che inizia dal pin e finisce al puntatore del mouse. Puntate ora al terminale invertente dell'op-amp e cliccate ancora una volta con il pulsante sinistro. ISIS considera l'ultimo pin come quello finale del collegamento ed invoca *Wire Auto Router* (WAR) per deciderne il percorso più appropriato. Fate la stessa cosa per ciascun terminale di R1 seguendo lo schema elettrico. Provate inoltre ad evidenziare un componente per poi spostarlo, osservando come WAR riposiziona i collegamenti in accordo alla nuova posizione dei componenti stessi.
Se non siete soddisfatti del percorso che WAR ha scelto per il collegamento, potete anche editarlo manualmente.
Per farlo, evidenziate il collegamento (il concetto è simile ad ogni oggetto evidenziato di ISIS, quindi puntate il collegamento e cliccate il pulsante destro del mouse) provando poi a trascinarlo inizialmente dagli angoli e quindi dalla posizione mediana.
Se volete collegare manualmente (evitando WAR), potete semplicemente cliccare con il pulsante sinistro sul primo pin e ancora per ciascun punto, lungo il percorso, in cui volete un vertice, e infine terminare cliccando sul secondo pin da collegare. Dopo un po' di pratica, avrete la netta sensazione di quando WAR può essere usato utilmente e quando, viceversa, è più utile evitarlo.
Per completare la prima sezione del circuito, dovrete ancora piazzare e collegare alcuni terminali. Nello specifico, abbiamo bisogno di due terminali generici, una massa e un terminale di alimentazione. Selezionate l'icona *terminali* - l'*Object Selector* dovrebbe mostrare l'elenco dei terminali disponibili come indicato in figura 5.
Ora posizioneremo e cableremo l'alimentazione dell'Op-amp. Selezionate il terminale *Power*, assicurandovi che esso sia correttamente orientato osservandone l'anteprima, e posizionatelo esattamente sopra il pin 7 del 741. Editate il terminale (click destro per evidenziarlo e quindi click sinistro per editarlo), digitare "+12V" nel campo *Stringa* ed infine premere ok.

Figura 5 *L'icona Terminali è selezionata e l'Object Selector indica i terminali disponibili*

Collegate il terminale al pin "7" dell'op-amp e provate a ripetere il procedimento per l'alimentazione negativa usando un altro terminale *Power* (ruotato stavolta di 180°), etichettato con "-12V" ed infine collegato al pin 4 dell'op-amp. Ora, posizionate il terminale *Ground*, assicurandovi che la sua anteprima sia correttamente orientata, e posizionatelo esattamente sotto R1. Selezionate il terminale *Default* dalla lista e posizionate due terminali come da schema, etichettandoli come "DAC1" e "DAC2". Infine collegate il terminale di massa al pin inferiore di R2 e i due terminali *Default* ai vertici dei collegamenti che entrano nell'op-amp. ISIS posizionerà le giunzioni quando necessario automaticamente, verificando che tre collegamenti si incontrano in questi punti.

ETICHETTE

Avrete notato che tutte le parti posizionate finora sullo schema hanno un numero di riferimento unico ed un valore. Il riferimento unico è impostato da una caratteristica di ISIS chiamata *Annotazione in tempo reale* che può essere trovata nel Menu *Strumenti* e che risulta abilitata per default.
Quando abilitata, essa annota i componenti così come questi vengono posizionati nello schema, risparmiando tempo e sforzo nel farlo manualmente.
E' possibile controllare sia la posizione che la visibilità di tali etichette - potete cambiarne i valori, spostarne la posizione o nasconderle se reputate non sia necessario visualizzarle.
La discussione che segue spiega in dettaglio come manipolare le etichette di un particolare componente.

Editare le Etichette dei Componenti

Se ingrandite uno dei resistori posizionati sullo schema, noterete che ISIS lo ha etichettato con un riferimento di parte unico (es. "R1") ed anche con un Valore (es."1k"). Potete editare entrambi i valori e la visibilità di tali campi tramite la finestra di dialogo *Modifica Componente*. A tale scopo selezionate l'icona *Modifica* e cliccate con il pulsante sinistro su uno dei resistori. Appare una finestra di dialogo mediante la quale potete editare sia il numero di riferimento parte che, come in questo caso, il valore di resistenza.
Di maggior interesse sono comunque le opzioni sulla visibilità. Può essere utile, specialmente in schemi ad alta densità di componenti, nascondere alcuni dettagli a lato dei componenti stessi - il compromesso è che dovrete poi editare le parti per vederne sia il riferimento che il valore.

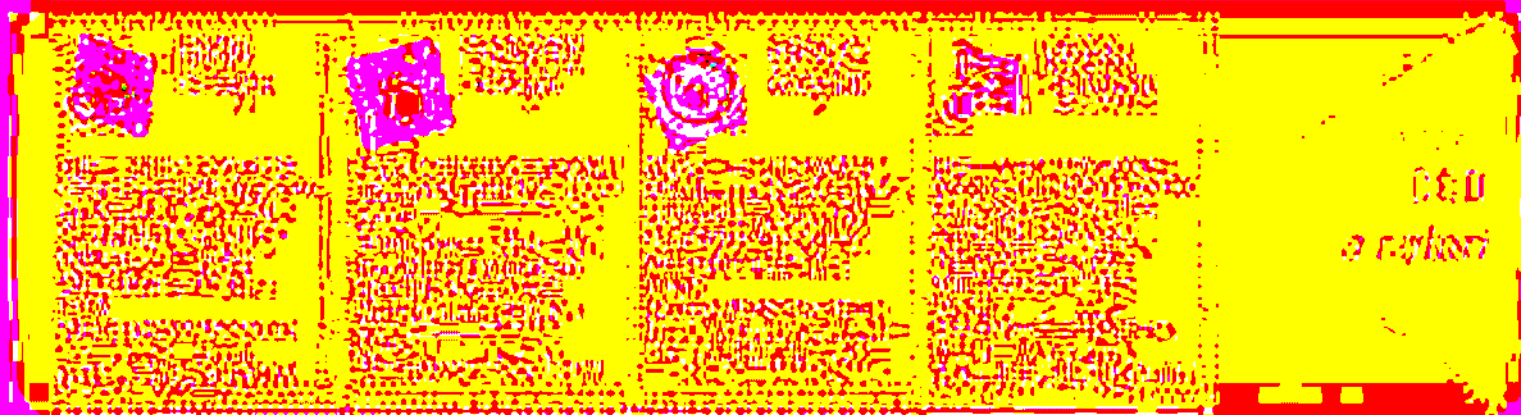
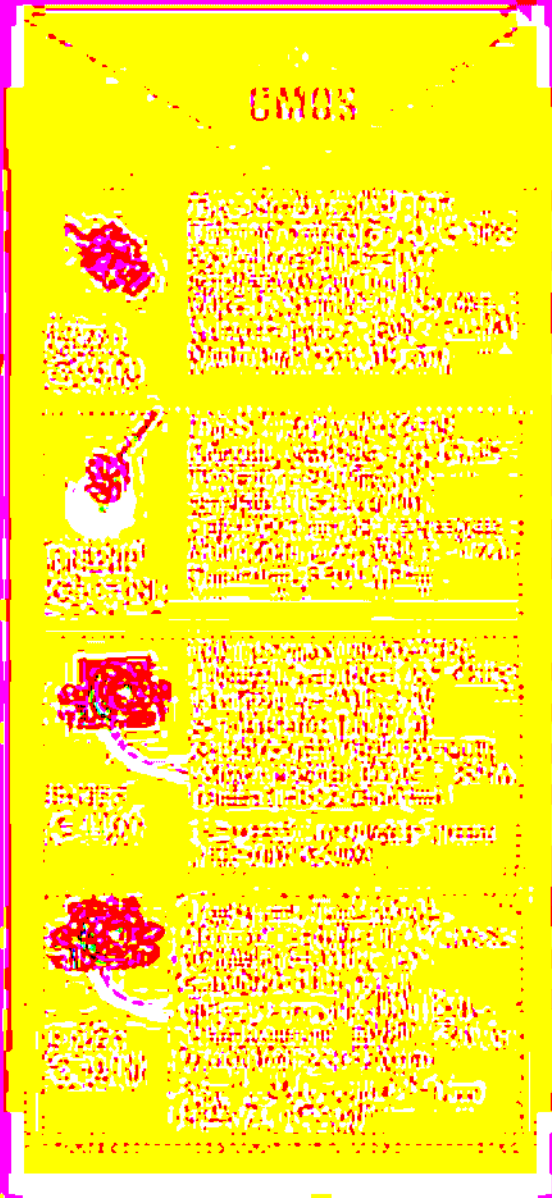
Spostamento di Etichette

Oltre alla possibilità di nascondere le etichette delle parti, potete anche spostarle all'occorrenza. Ciò può accadere quando una connessione deve posizionarsi laddove è ubicata una etichetta. Noi proveremo tale eventualità con le etichette "U1" e "741", spostandole in accordo allo schema di riferimento.
Il modo più semplice per spostare le etichette dell'op-amp è quello di modificare il cosiddetto valore di snap.
Muovendo il puntatore del mouse nell'area di *Editing*, avrete notato che le co-ordinate variano in passi discreti - 100 millesimi di pollice per default. Questo effetto è chiamato snap. Grazie allo snap potete posizionare i componenti ed altri oggetti in corrispondenza della griglia. La risoluzione dello snap può essere scelta con i comandi *Passo* nel Menu *Visualizza*, o direttamente tramite tastiera.
Premete quindi tasto F2 per aumentare la risoluzione dello snap e della griglia a 50th (il default è 100th) e quindi evidenziate l'op-amp.
Ora posizionate il puntatore sull'etichetta "U1" e con il pulsante sinistro sempre premuto, trascinatela nella sua corretta posizione sotto l'op-amp. Fate poi lo stesso con l'etichetta "741".
Quando avete terminato, riportate lo snap a 100th premendo F3.

Anche se il *Real Time Snap* di ISIS è in grado di localizzare pin e collegamenti posti non esattamente sulla griglia, lavorare in modo coerente con lo stesso valore di snap e griglia aiuta a tenere ordinati e puliti gli schemi.

EDIT DI INTERE SEZIONI CIRCUITALI

Avrete notato che la sezione di circuito che avete disegnato finora è localizzata al centro dello schermo, mentre essa dovrebbe essere nell'angolo in alto a sinistra.
Per spostare la sezione in quel punto occorre evidenziare tutti gli oggetti che avete posizionato tracciando intorno ad essi un rettangolo di selezione: a tale scopo puntare il primo vertice del rettangolo in alto a sinistra di tutti gli oggetti da spostare; premere e tenere premuto il pulsante destro del mouse e trascinare il puntatore fino al secondo vertice in basso a destra degli oggetti da selezionare. L'area selezionata è mostrata con un rettangolo color ciano (notate che il click destro iniziale riporta oggetti eventualmente già evidenziati al loro stato normale) e alla fine dell'operazione solamente quegli oggetti all'interno del rettangolo saranno evidenziati.
Ora cliccate con il pulsante sinistro sulla icona *Sposta Blocco* come mostrato in figura 6.
Appare un rettangolo blu che circonda le parti evidenziate che è possibile spostare in tutte le direzioni. Spostatelo verso l'angolo in alto a sinistra del foglio.
Il bordo del foglio appare in blu scuro, quindi potete riposizionare la sezione in modo tale che rimanga tutta nell'angolo in alto a sinistra del disegno. Cliccando il pulsante sinistro si effettua lo spostamento o, viceversa cliccando il pulsante destro del mouse, si termina l'operazione in corso.
Potreste aver notato che durante l'operazione di spostamento l'area di *Editing* scorre automaticamente se ciò occorre. Nel caso vogliate lo scorrimento in altri casi (quando, cioè, **non** siete in modalità di posizionamento o spostamento), potete sempre usare la

Figura 6 Le icone Modifica Blocchi con il comando Sposta Blocco selezionato

modalità *Shift Pan* (ovvero tendo premuto il tasto SHIFT e sbattendo il puntatore sui bordi dell'aera di editing).

LA PRATICA RENDE PERFETTI

Ora che avete acquisito i rudimenti, potete migliorare disegnando la sezione circuitale incentrata sull'op-amp U2:A e riportata in figura 7.
Iniziate selezionando dalla libreria di parti il condensatore e il TL074 usando i metodi discussi in precedenza. Per aiutarvi, diremo che usiamo dei condensatori ceramici nello schema, quindi cercate nella libreria un "*ceramic 220pF capacitor*". Usate le varie tecniche di editing che avete imparato fino ad ora in modo che tutti i componenti siano posizionati nel modo corretto.
Una volta terminata la sezione del filtro in modo soddisfacente, evidenziate l'intera sezione usando il procedimento noto e la funzione *Copia Blocco* per farne una seconda copia che posizionerete a destra. Notate che nel circuito di riferimento le due sezioni sono identiche ad eccezione di alcuni resistori. Selezionate quindi "MINRES12k" dall'Object Selector e spostate il puntatore del mouse sul resistore da sostituire di 15k, posto nella sezione di destra appena copiata. Premete il pulsante sinistro del mouse **dentro** il corpo del resistore esistente allineandone i pin - ciò sostituirà la MINRES15k esistente con la MINRES12k selezionata, mantenendo i collegamenti già in essere.
Potete fare la stessa cosa sostituendo il resistore da 100k con uno da 10k.
Ripetete tale procedimento con le due sezioni di filtro rimanenti poste in basso nel circuito di riferimento (rammentate di selezionare dalla libreria parti un "*1n5 Ceramic Capacitor*") in modo da avere quattro filtri in totale, corrispondenti, cioè, al circuito di riferimento usato nel presente tutorial. Come in precedenza, allo scopo di conformarvi al circuito di riferimento, dovrete sostituire nelle nuove sezioni appena copiate i resistori aventi valore diverso.
Completati e posizionati i quattro filtri, ultimate i collegamenti tra loro e posizionate sullo schema un deviatore SW-SPDT (SW1).

Figura 7 *La prima sezione del filtro del circuito del tutorial*

ANNOTARE UNO SCHEMA

ISIS fornisce quattro possibili modi per annotare (denominare) componenti:

- *Annotazione Manuale* - È il metodo che avete già usato per etichettare il primo op-amp e il resistore. Qualsiasi oggetto può essere editato sia selezionando l'icona *Modifica* cliccandovi sopra con il pulsante sinistro del mouse, che cliccando prima con il tasto destro (evidenzia) e poi con quello sinistro nella modalità standard di posizionamento.
  Qualunque modo voi scegliate, apparirà una finestra di dialogo che userete per introdurre le relative proprietà come *Riferimento parte, Valore* e così via.
- *Strumento Assegnazione proprietà* (PAT) - Questa opzione permette di generare etichette per qualsiasi oggetto di ISIS e consente anche di generare sequenze di etichette sia nel foglio di lavoro attuale che in tutti i fogli del progetto.
- *Annotazione Automatica* - permette di annotare automaticamente il circuito più complesso, in un attimo. L'opzione considera naturalmente anche quelle parti a sezione multipla come ad esempio la porta TTL NAND 7400, annotando le varie porte in modo opportuno.
- *Annotation Real Time* - Tale caratteristica, quando abilitata, annoterà i componenti in tempo reale nello stesso momento in cui questi sono posizionati nello schema, ovviando alla necessità di farlo manualmente in seguito. Comunque, come per l'*Annotazione Automatica*, tale procedimento non è affatto interattivo, e quindi, non offre all'utente alcun controllo. Se desiderato, la caratteristica di *Annotazione Real Time* può essere disabi-

litata, oppure riabilitata con il comando *Annotazione in tempo reale* nel *Menu Strumenti* (CTRL+ N per default).

Durante l'uso pratico, si usa normalmente l'opzione *Annotazione Real Time* abilitata da default (come abbiamo fatto in questo tutorial), ed in secondo momento si può usare il comando *Annotazione Automatica* oppure lo *Strumento Assegnazione proprietà* per cambiare l'annotazione delle parti per ragioni estetiche di disegno. L'annotazione manuale è normalmente usata solo nel caso di parti aventi sezioni eterogenee multiple (come ad esempio un relay e due contatti) dove avete necessità di indicare esplicitamente ad ISIS quale contatto appartiene a quel determinato relay.

Strumento di Assegnazione Proprietà (PAT)

Supponiamo di voler annotare alcuni resistori da R5 in avanti usando il comando PAT. Avremo quindi bisogno di generare una sequenza che inizia da R5 in poi (R5, R6, R7, ecc.). Selezioniamo allora l'opzione *Strumento Assegnazione proprietà* nel *Menu Strumenti*. Nel campo *Stringa*, digitare **REF=R**, quindi spostiamo il cursore sul campo successivo Valore e digitiamo il valore 5. Assicuriamoci che l'opzione *Al Click* sia selezionata ed infine clicchiamo sul pulsante OK oppure premiamo ENTER. Ogni volta che PAT assegna una stringa all'oggetto da annotare, il carattere cancelletto ("#") nel campo *Stringa* sarà sostituito dall'attuale valore digitato nel campo *Valore* e successivamente questo sarà automaticamente incrementato. Ciò è mostrato nella figura 8.
ISIS imposta automaticamente la modalità *Modifica* in modo che possiate annotare la parte cliccando semplicemente con il pulsante sinistro del mouse su di essa. Puntate ad esempio su R7 e premete il pulsante sinistro del mouse. PAT annota il resistore in R5 e la parte è ridisegnata. Fate lo stesso con il vecchio resistore denominato ancora R5 e notate come il campo *Valore* del PAT si incrementa and annota il resistore in R6. Potete ora annotare i resistori rimanenti. Provate questo strumento fintanto che non vi è chiara la modalità di funzionamento - anche se un po' macchinoso all'inizio, PAT è uno strumento estremamente potente che eliminerà gran parte delle difficoltà durante l'editing degli schemi.

Figura 8 Lo Strumento di Assegnazione Proprietà impostato per riannotare i resistori che iniziano da R5

Annotazione Automatica

ISIS fornisce l'opzione di annotazione automatica che deciderà per voi i riferimenti unici di parte dei componenti nel vostro schema. Esso può essere usato per annotare tutti i componenti o solo quelli che non lo sono ancora - quelli, cioè, con un "?" nel campo *riferimento*.
Lo strumento di Annotazione Automatica ha due modalità operative:
Incrementale: ha effetto su quei componenti (progetto o foglio attuale) che non sono stati ancora annotati. Totale: Tutti i componenti (progetto o foglio attuale). Tutti i componenti, eccetto quelli aventi sezioni eterogenee multiple saranno annotati. Poiché non abbiamo parti di questo tipo nel nostro schema useremo lo strumento *Annotazione Automatica* in modalità Totale. Per far ciò, chiamiamo il comando *Annotazioni* nel Menu *Strumenti*, selezioniamo l'opzione *Totale*, e finalmente clicchiamo su OK. Lo schema sarà ridisegnato e rifletterà le modifiche appena effettuate.

NEL PROSSIMO NUMERO

Nel prossimo numero ci occuperemo della creazione di nuovi dispositivi, della definizione dei loro terminali, e dell'associazione ad uno o più contenitori (package) da usare nella realizzazione del circuito stampato. Nei numeri successivi vedremo come realizzare un PCB utilizzando gli strumenti di piazzamento e sbroglio automatici, come estrarre i file gerber e vedremo come simulare un circuito utilizzando il VSM di Proteus.

Alimentatori switching:

Vedremo in questa puntata come realizzare con estrema semplicità alimentatori multi-uscita. Per la parte teorica, svilupperemo adeguatamente il concetto dell'isolamento ingresso-uscita, fondamentale negli alimentatori off-line. Infine, sfrutteremo quanto detto per realizzare un interessante alimentatore a 3 uscite, +5V, +12V, -12V. Buona lettura!

GLI AVVOLGIMENTI: APPROFONDIMENTO

Sappiamo già che la quasi totalità degli avvolgimenti dei trasformatori e delle induttanze sono in rame, isolato esternamente per mezzo di una sottile "patina" di vernice: da qui il nome comune di rame "smaltato".
L'isolamento è quasi sempre necessario per gli avvolgimenti per switching, in quanto per motivi di induttanza si avvolgono in generale **più strati sovrapposti e ravvicinati:** se il rame non fosse isolato, si verrebbe a creare un cortocircuito che annullerebbe ogni nostro tentativo di realizzare un'induttanza.
Dalle formule già viste in precedenza, sapete anche che **la resistenza di un filo di rame è inversamente proporzionale alla sua sezione**, cioè più il filo è "grosso", minore è la sua resistenza.
Sapete anche che, in un filo, la potenza dissipata in calore dipende proprio dalla sua resistenza, ne consegue quindi che:

Tanto maggiore è la sezione (proporzionale al diametro) di un filo di rame, tanto maggiore sarà la sua capacità di trasporto di corrente (a parità di potenza dissipata).

Come scegliere, in maniera analitica/matematica, il giusto diametro del filo per i nostri avvolgimenti? In genere si utilizza la **densità di corrente**, che è semplicemente data dalla corrente totale divisa per la sezione:

$$d = I / S$$

d in A/m², I in Ampere, S in m² (sezione)

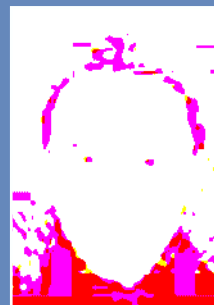
La densità di corrente deve essere minore oppure uguale ad un limite standard.

Sul Web potete trovare molti riferimenti, a volte in contrasto tra loro, e a volte espressi in unità di misura non standard, cioè non appratenenti al Sistema Internazionale. Ad esempio si trovano spesso i valori di **500 e 700 *circular mils* per Ampere RMS**. Osservando il riquadro di approfondimento per la definizione dei circular mils,

Circular mils

È un'unità di misura della sezione di un filo. Rappresenta la sezione (area o superficie) di un filo avente diametro pari a 1mil. **1mil corrisponde a 1/1000 di pollice**, e vale 1 pollice = 25,4mm.
Quindi 1mil = 25,4 / 1000 = 0,0254mm, e applicando le note regole dell'area di un cerchio, otteniamo infine:

$$\textbf{1 circular mil} = (0{,}0254/2)^2 * \pi = = 5{,}067e\text{-}4\,mm^2$$

Switching flyback multi-uscita

di Romano Bernarducci
r.bernarducci@farelettronica.com

diventa facile calcolare la densità di corrente consigliata:

$$d = 500 * 5{,}067e\text{-}4 = 0{,}253\ mm^2/Ampere$$
(500 circular mils/Arms)

oppure

$$d = 700 * 5{,}067e\text{-}4 = 0{,}355\ mm^2/Ampere$$
(700 circular mils/Arms)

Il secondo valore è estremamente conservativo: onde evitare grossi avvolgimenti noi useremo la prima delle due espressioni precedenti.
In pratica questa espressione dice che per ogni Ampere di corrente che vogliamo far scorrere nell'avvolgimento, dobbiamo avere almeno 0,253 mm^2 di sezione di filo. Alternativamente si può affermare che, attraverso un filo di sezione pari a 1mm^2, possiamo far scorrere una corrente massima di 1 / 0,253 = 3,947A. La tabella 1 fornisce i valori di corrente massima, a seconda della sezione del filo in mm^2, per entrambe le due densità citate, quella standard e quella più "prudente".

ISOLAMENTO INGRESSO-USCITA: CONSIDERAZIONI AVANZATE

Nella scorsa puntata ho appena accennato all'estrema importanza dell'isolamento tra le tensioni di ingresso e di uscita, in un alimentatore switching. L'isolamento diventa assolutamente indispensabile, in applicazioni consumer (es. DVD, televisori, PC) quando il nostro alimentatore è del tipo *off-line*, cioè collegato direttamente alla rete elettrica.

Esaminiamo in figura 1 lo schema di principio di uno switching *off-line*, **in topologia flyback**. Per prima cosa, incontriamo sulla sinistra un classico rettificatore a doppia semionda, seguito dal condensatore di livellamento che trasforma la tensione pulsante a 100Hz in una quasi-conti-

Diametro filo [mm]	Sezione filo [mm^2]	Corrente max "standard" [Arms]	Corrente max "prudente" [Arms]
0,10	0,008	0,03	0,02
0,20	0,031	0,12	0,09
0,30	0,071	0,28	0,20
0,40	0,126	0,50	0,35
0,50	0,196	0,78	0,55
0,60	0,283	1,12	0,80
0,70	0,385	1,52	1,08
0,80	0,503	1,99	1,42
0,90	0,636	2,51	1,79
1,00	0,785	3,1	2,21
1,50	1,767	6,98	4,98
2,00	3,142	12,42	8,85

Tabella 1 *Corrente massima in un filo di diametro noto*

nua. Calcoliamo la tensione quasi-continua disponibile, supponendo una tensione di rete nominale di 230VAC. Ai neofiti ricordo che per valore nominale della tensione di rete, si intende sempre la **tensione alternata efficace o RMS.** Per ottenere la tensione di picco, occorre moltiplicare la tensione efficace per una costante pari alla radice quadrata di 2, che vale circa 1,4142. Otteniamo quindi:

$$V_{PICCO,NOM} = V_{RMS} * 1,4142 =$$
$$= 230 * 1,4142 = 325VDC$$

che rappresenta il valore nominale dell'alta tensione continua, presente sul collegamento evidenziato in rosso in figura 1. Ho volutamente trascurato la caduta di tensione sui diodi del ponte, che vale $2 * V_D = 2V$ circa, ininfluenti rispetto a 325. Notate che per la tensione alternata di rete, è buona norma considerare una tolleranza di **+-15%** rispetto al valore nominale. Nel caso peggiore avremo quindi sul condensatore di ingresso, una tensione continua pari a:

$$V_{PICCO,MAX} = V_{RMS} * 1,15 * 1,4142 =$$
$$= 230 * 1,15 * 1,4142 = 374VDC$$

Ricordo che il fattore moltiplicativo 1,15 rappresenta l'incremento percentuale del 15% della VAC nominale. Quindi:

la tensione di lavoro del condensatore di ingresso in uno switching *off-line* per la tensione di rete nominale di 230VAC, deve essere pari ad almeno **375V**.
Per garantire un certo margine di sicurezza ed incrementare la vita operativa del condensatore, è comune scegliere tensioni di lavoro di **400V**.

Dopo questa necessaria premessa, riprendiamo la figura 1. Facciamo un elenco dei **vincoli** che un tale schema di principio, seppur semplice, si porta appresso:

1. Il controller (ad esempio di tipo PWM) deve essere alimentato per poter funzionare correttamente. Sinora il controller è stato sempre alimentato dalla tensione di ingresso. **Purtroppo in questo caso abbiamo a disposizione una tensione di ingresso enormemente più alta di quella massima di funzionamento dei circuiti integrati.** Quest'ultima vale di norma 30/40V, e solo in casi particolari può arrivare a 60V o più. **È necessario quindi generare una "bassa"**

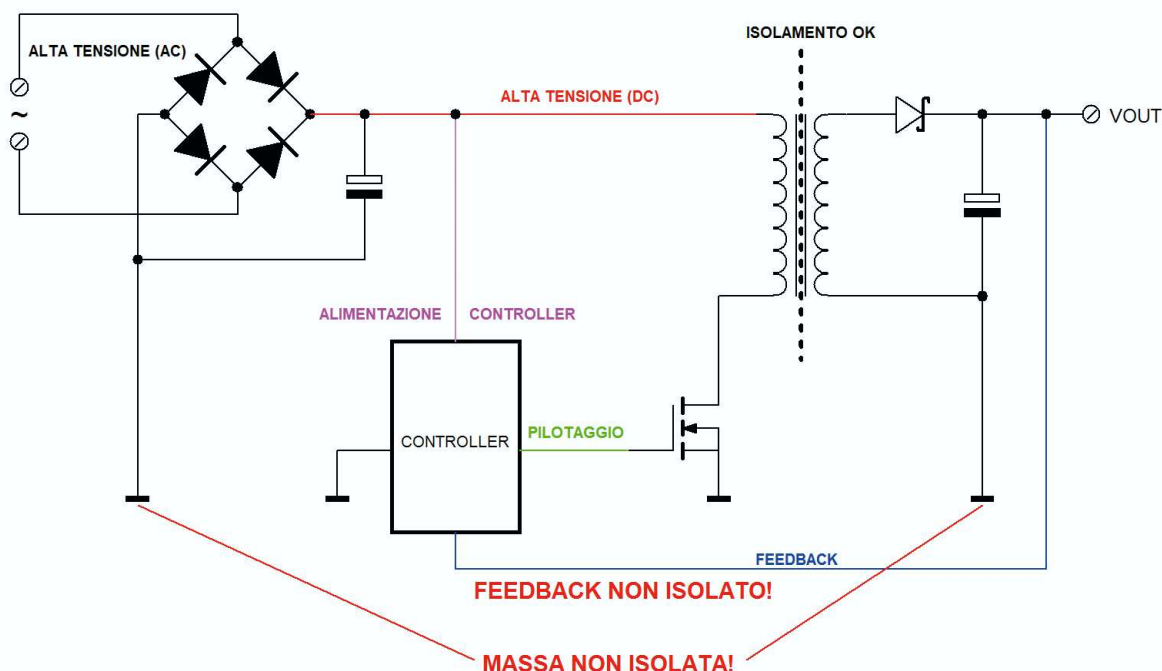



Figura 1 *Schema di principio di uno switching off-line non isolato*

tensione per alimentare il controller.

2. Il controller deve poter "leggere" la tensione **continua** di uscita, per poter regolare, tramite controreazione, il duty-cyle del PWM di comando.
3. **La massa della tensione di uscita NON deve essere collegata elettricamente alla massa di ingresso**, poiché quest'ultima è collegata alla rete elettrica.

Soddisfare contemporaneamente tutti questi vincoli non è cosa semplicissima... potete formulare le vostre ipotesi prima di andare avanti.

Vincolo 1:
supponiamo che il controller PWM assorba 10mA per il suo funzionamento interno, a cui occorre aggiungere la corrente necessaria per caricare/scaricare, alla frequenza di switching, il gate del MOSFET.
Ipotizziamo a spanne un consumo totale di 40mA, alla tensione di funzionamento tipica del controller, pari ad esempio a V_{CC} = 20V.
Se avete pensato ad una semplice resistenza tra l'alta tensione (375V) e il controller...la risposta è NO! Vediamo perchè applicando la legge di Ohm:

$$R = (V_{IN} - V_{CC}) / I = (375 - 20) / 0,040 = 8875\Omega$$

Sembra tutto ok... calcoliamo ora la potenza dissipata:

$$P_{DISS,R} = I^2 * R = 0,040^2 * 8875 = 14,2W!!$$

Assolutamente non accettabile, a meno di non voler costruire una..stufa!!
Ok, possiamo costruire un alimentatore di supporto, a bassa potenza, di tipo lineare per semplicità, che fornisca i 20V a bassa corrente, come in figura 2.

Questa soluzione viene realmente impiegata

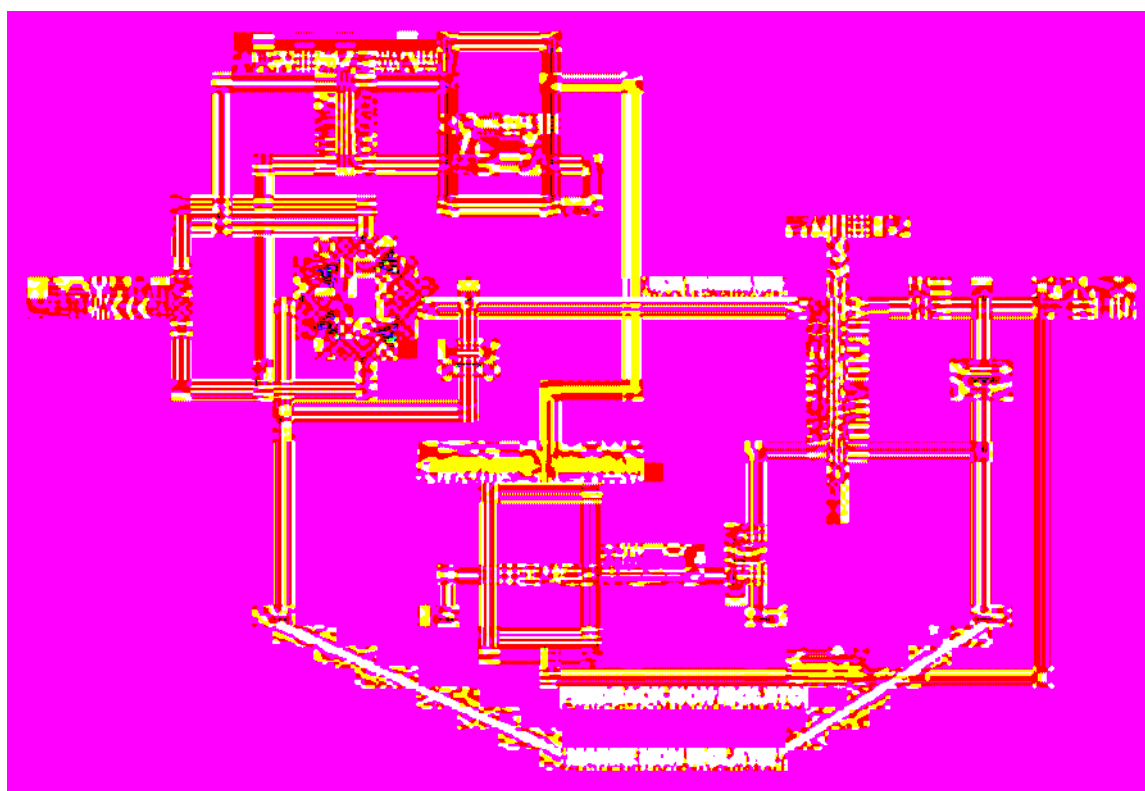



Figura 2 *Schema di principio di uno switching off-line non isolato, con alimentatore ausiliario*

in alcuni alimentatori di alta potenza (dell'ordine dei kW), in quanto torna utile avere una tensione di servizio, sempre disponibile anche quando lo switching principale è spento. **Tuttavia non è ancora accettabile in un alimentatore "consumer", per i quali il costo finale del prodotto è determinante.** Ciò non diminuisce la sua validità per i vostri alimentatori hobbistici... vedremo comunque più avanti altre alternative.

Vincoli 2 e 3:
Questi due vincoli sono contrastanti...vogliamo che le masse siano isolate **e contemporaneamente** riportare al controller l'informazione sull'entità della tensione CONTINUA di uscita.
In generale, l'isolamento galvanico può essere ottenuto in due modi:

1. Tramite un trasformatore di accoppiamento, che trasporta "l'informazione" dal primario al secondario, tramite il campo magnetico.
2. Tramite un optoisolatore, che sfrutta invece la luce emessa da un LED e captata da un fotoricevitore (vedi riquadro di approfondimento).

Per la sua costituzione, il trasformatore può solo trasportare un segnale in AC, e questo lo esclude immediatamente dal novero delle possibilità, per gli schemi di figura 1 e 2. L'unica possibilità rimasta è l'optoisolatore, ed infatti è quello che si usa normalmente, come in figura 3.

Cosa è un optoisolatore

È un dispositivo costituito da una sorgente luminosa a LED, in genere infrarosso (quindi non visibile), che emette verso un dispositivo fotosensibile, come un fotodiodo, un fototransistor, eccetera. In genere i due dispositivi sono inglobati in un contenitore plastico tipo circuito integrato, spesso di colore bianco. Poiché lo scopo principale di un optoisolatore è quello di ISOLARE, il LED ed il fotosensore si trovano sempre in posizioni opposte e mai adiacenti, in modo da aumentare la loro separazione fisica al massimo.

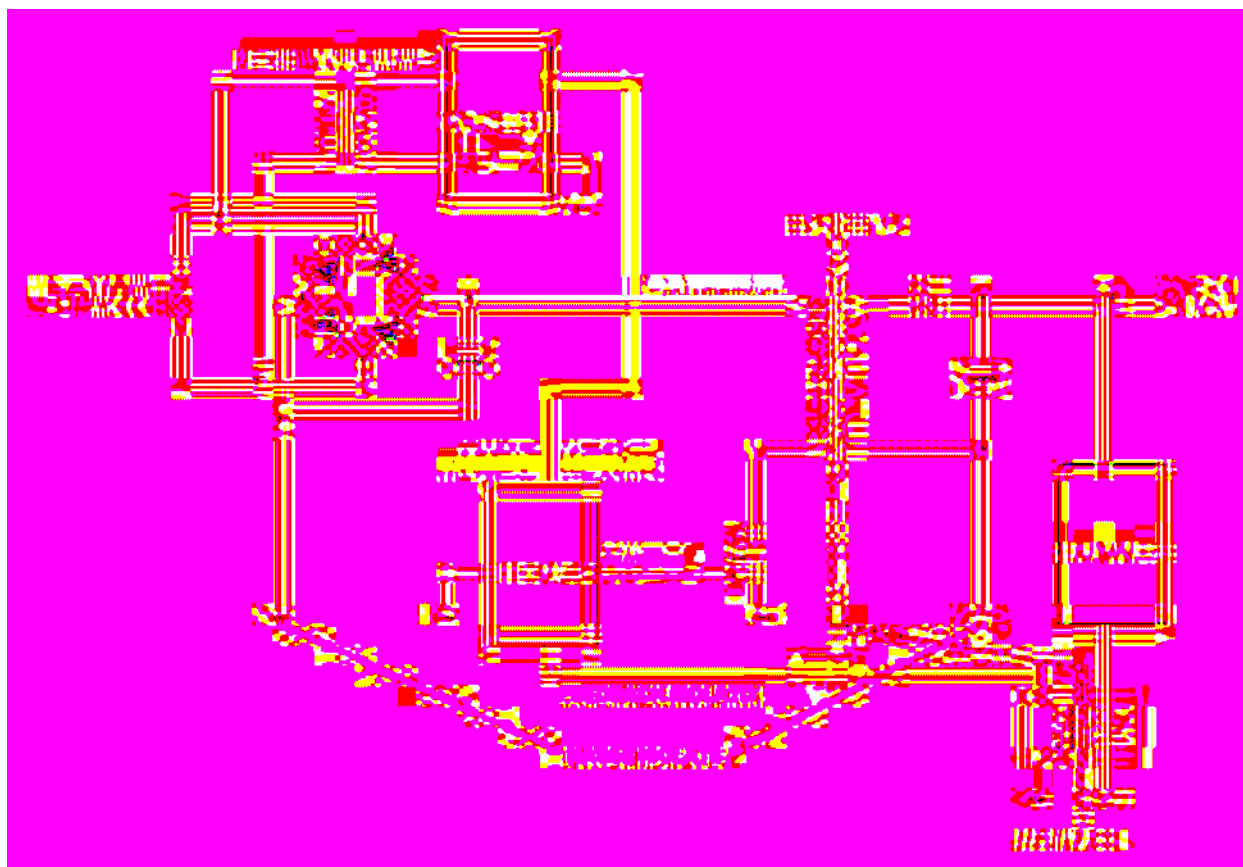
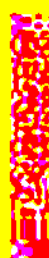
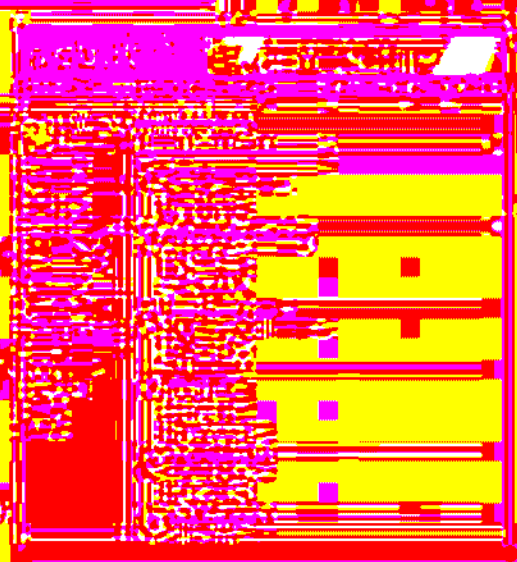
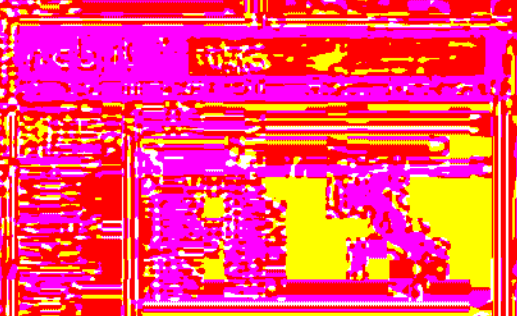



Figura 3 *Schema di principio di uno switching off-line ISOLATO, con alimentatore ausiliario*

inserimento
gratuito

Il LED interno all'optoisolatore viene attivato dal comparatore quando la tensione di uscita supera un valore predefinito. La luce emessa, colpendo il dispositivo fotosensibile, lo attiva.
Nel caso di figura 3, il fototransistor NPN chiuderà verso massa il piedino di feedback del controller, trasferendogli così l'informazione sul livello della tensione di uscita.

NOTA
L'inserimento dell'optoisolatore nell'anello di feedback, altera considerevolmente i calcoli sulla stabilità, che diventano ancora più complicati rispetto al caso standard, senza optoisolatore.
Per i più esperti segnalo che trattazioni analitiche sono reperibili sul Web.

A questo punto abbiamo soddisfatto tutti i vincoli precedenti, ma resta il problema della scarsa ottimizzazione... Come fare?
Beh, pensate che, durante il funzionamento, avete a disposizione delle bellissime oscillazioni sul primario del trasformatore... perchè non utilizzarle per realizzare l'alimentatore ausiliario?
È quello che viene fatto in figura 4.

Come vedete, basta aggiungere un altro avvolgimento dal lato del primario (che in realtà si comporta come un secondario, cioè si attiva quando il primario è OFF, notate i "pallini"), il quale durante il funzionamento fornisce dei picchi di tensione, che rettificati e livellati da un condensatore, alimentano il controller.
Questo avvolgimento, per ovvi motivi, viene spesso indicato come **avvolgimento ausiliario** (*auxiliary winding* in inglese).
Tutto ok?
Ebbene, ancora no! Ci si presenta la versione elettronica del dilemma dell'uovo e della gallina: il controller ha bisogno di oscillare per generare la tensione che serve a farlo funzionare! Niente paura, il problema viene risolto brillantemente con un piccolo aiuto "esterno", come in figura 5.

L'aiuto è costituito dalla resistenza *Rstart*, indicata nel cerchio rosso... Confusi?
A differenza di quanto discusso a proposito del Vincolo 1, dove avevamo mostrato che una soluzione siffatta, con una resistenza verso l'alta tensione, era impraticabile, **ora la resistenza può avere un valore molto più elevato (quindi poca potenza dissipata)**, in quanto deve solo fornire la poca corrente necessaria alle prime oscillazioni. **Una volta partita l'oscillazione primaria, l'energia necessaria per il funzionamento del con-**

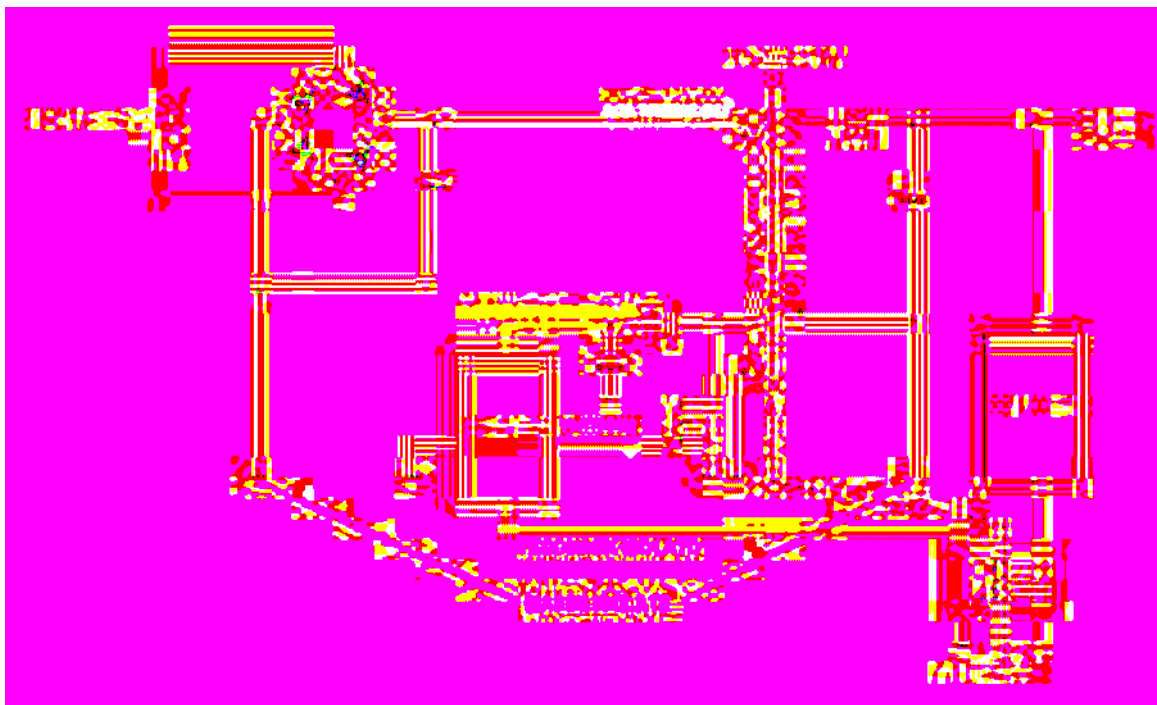



Figura 4 *Schema di principio di uno switching off-line ISOLATO, SENZA alimentatore ausiliario*

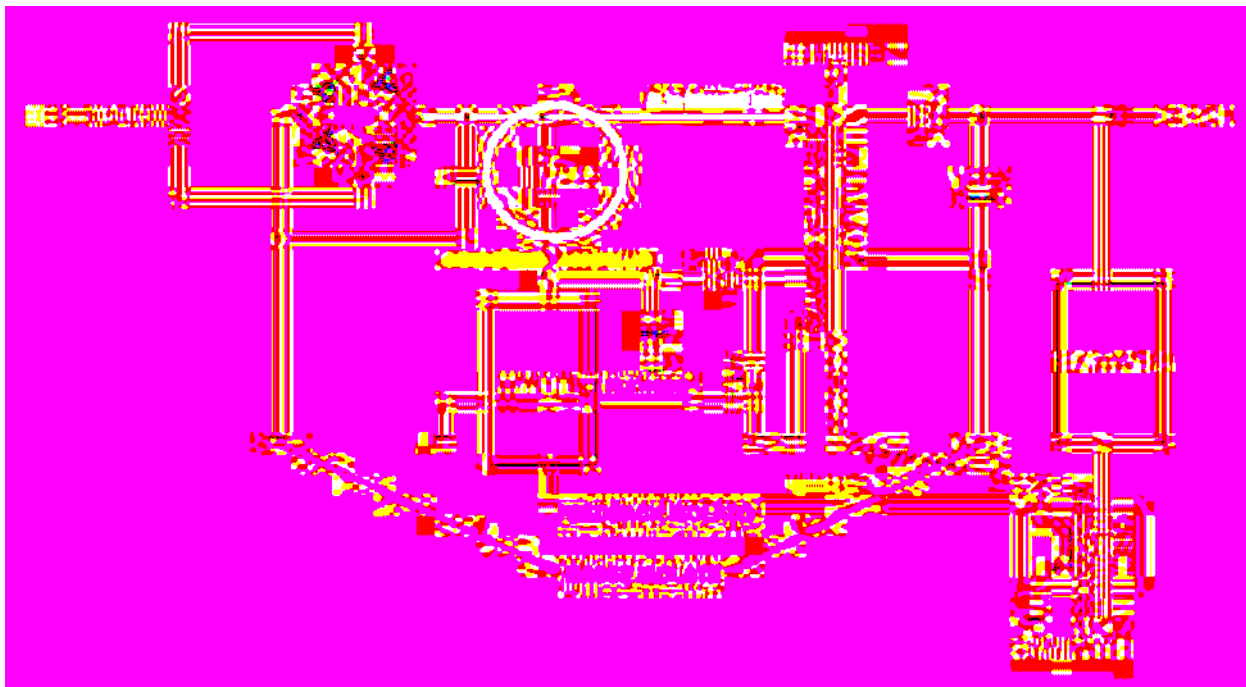



Figura 5 Schema di principio (I) di uno switching off-line ISOLATO

Le so tutte!!!

Risposta al quesito codice LST24307

Per quanto riguarda le tensioni, si nota che la V_{OLmax} della porta standard è minore della V_{ILmax} della porta LS quindi si è sicuri che l'uscita bassa della porta standard viene correttamente interpretata dalla porta LS. Sul livello alto si ha una situazione analoga: la V_{OHmin} della porta standard è maggiore della V_{IHmin} della porta LS e ciò garantisce il corretto funzionamento anche sul livello alto. Analizziamo ora la situazione relativa alle correnti ipotizzando di pilotare N porte LS. Sul livello alto la corrente assorbita dalle porte LS sarà N volte quella assorbita da una singola porta quindi $NI_{IHmax(LS)}$. Questa corrente dovrà essere minore di quella massima erogabile dalla porta standard quindi dovrà essere:

$NI_{IHmax(LS)}<I_{OHmax(SN)}$ da cui si ricava $N<I_{OHmax(SN)}/I_{IHmax(LS)}=10$

Analogamente, sul livello basso dovrà essere $NI_{ILmax(LS)}<I_{OLmax(SN)}$ da cui $N<I_{OLmax(SN)}/I_{ILmax(LS)}=5$.
Dai risultati ottenuti si evince che la porta standard può pilotare correttamente fino a 5 porte LS.

Il vincitore di LUGLIO/AGOSTO 2005 (Vincitore LST24106)
I nostri complimenti a Claudio Mazzurco di Misterbianco (CT)
che vince un abbonamento a Fare Elettronica!

troller verrà fornita quasi interamente dall'avvolgimento ausiliario. Considerate *Rstart* come un motorino di avviamento! Un ultima considerazione: non mi stancherò mai di ripetere che, nel mercato consumer, ogni centesimo di euro/dollaro conta... come fare per ottimizzare ancora lo schema precedente? Considerate che l'avvolgimento ausiliario è in realtà un secondario, proprio come quello che genera V_{OUT} (solo che è dalla parte del primario). Per le note proprietà dei trasformatori, le tensioni di picco presenti sull'avvolgimento ausiliario e sul secondario di uscita sono proporzionali, con una costante di proporzionalità che dipende dal rapporto spire... allora **perchè non usare la tensione fornita dall'avvolgimento ausiliario, come "facsimile" di quella di uscita?** ***In altre parole, misurando la prima tensione, posso "stimare" con buona precisione la seconda.*** È quello che viene fatto in figura 6.

Come si vede la tensione di feedback viene prelevata direttamente dalla tensione di alimentazione del controller!

È da notare che la precisione della tensione di uscita V_{OUT}, in questo caso, è nettamente inferiore rispetto allo schema di figura 5. Posso affermare con buona certezza che la stragrande maggioranza degli switching *off-line*, di **medio-bassa potenza (max. circa 150W)**, presenta uno schema di principio che ricalca fedelmente una delle figura 5 o 6.

NOTA

La discussione sin qui fatta sulla necessità dell'isolamento, non si applica in alcuni casi: per fare un esempio, per uno switching off-line regolabile che alimenti un ventilatore interno ad una apparecchiatura, inaccessibile dall'utente durante il funzionamento normale.

ALIMENTATORI MULTI-USCITA

Uno dei motivi della larghissima diffusione degli alimentatori flyback, è legata all'estrema facilità con cui si riescono ad implementare switching ad uscite multiple. Infatti, se ricordate la

Importante

Mi auguro che tra i miei lettori ci siano anche molti neofiti. Particolarmente a questi, ma non solo, mi permetto di fornire i sequenti consigli:

Non eseguite prove o misure su switching *off-line* se non siete più che esperti negli aspetti di sicurezza elettrica.

I condensatori di filtro possono restare carichi anche per diversi secondi dopo lo spegnimento del dispositivo, e l'alta tensione residua può risultare mortale.

Ricordate sempre di **interrompere ambedue i collegamenti** dello switching alla rete (interruttore doppio). Collegate una spia luminosa a valle dell'interruttore, in modo da avere l'immediato feedback visivo dello stato di alimentazione dello switching.

Ricordate che non potete collegare un oscilloscopio direttamente all'ingresso dello switching *off-line* (ad esempio per osservare le forme d'onda sui pin del controller), perchè in genere **lo chassis dell'oscilloscopio, che di norma è a terra, coincide con la massa della sonda** (provate con un tester).

Questo causerebbe un corto-circuito tra fase e terra con intervento del differenziale...se c'è!.

Per evitare questo problema, usate sempre **un trasformatore di isolamento(*)** per lo switching o per l'oscilloscopio: **ciò non vi esime dal prestare comunque la MASSIMA ATTENZIONE.**

(*) Trasformatore con rapporto spire tra primario e secondario pari a 1. **Non altera quindi il valore della tensione di rete, ma isola galvanicamente il carico.** Vari modelli, che si differenziano in genere per la potenza disponibile, sono reperibili sui cataloghi RS e Distrelec.

puntata precedente, il secondario del flyback viene attraversato da corrente solo quando il primario è OFF. In parole semplici, durante la fase ON, la "pompa" del primario carica il serbatoio magnetico, poi si scollega e subito dopo il serbatoio si "svuota" sul secondario. Supponiamo che il controller regoli alla perfezione una tensione di 5V sul secondario numero 1, composto da 10 spire. **È evidente che, se avvolgiamo un altro secondario da 20 spire sullo stesso nucleo, vale la regola dei trasformatori: ai suoi capi comparirà il doppio della tensione del primo, cioè 10V.**

È normale che la regolazione (stabilità alle variazioni del carico, ad esempio) delle uscite non controllate sarà molto peggiore dell'unica uscita controllata; tuttavia nella maggioranza dei casi questo non è un problema. Nell'alimentatore triplo che presenterò più avanti, la tensione di 5V, usata ad esempio per alimentare un microprocessore, sarà regolata dal controller, mentre le altre due tensioni di +-12V potranno servire per alimentare degli op-amp, che non richiedono notoriamente un'elevata stabilità.

"FLYBACK" MULTIUSCITA: PARAMETRI DI PROGETTO

Supponiamo di voler realizzare un alimentatore switching, di tipo *"flyback"*, multiuscita, avente le seguenti caratteristiche:

$V_{IN,NOM}$ = 24VDC ±25%, cioè:
$V_{IN,MIN}$ = 18VDC
$V_{IN,MAX}$ = 30VDC

V_{OUT1} = 5VDC
$I_{OUT1,MAX}$ = 1A

V_{OUT2} = 12VDC
$I_{OUT2,MAX}$ = 0,2A

V_{OUT3} = -12VDC
$I_{OUT3,MAX}$ = 0,2A

TIPOLOGIA "FLYBACK" MULTIUSCITA: CALCOLI

1. Calcoliamo la potenza di uscita massima per ciascuna uscita:

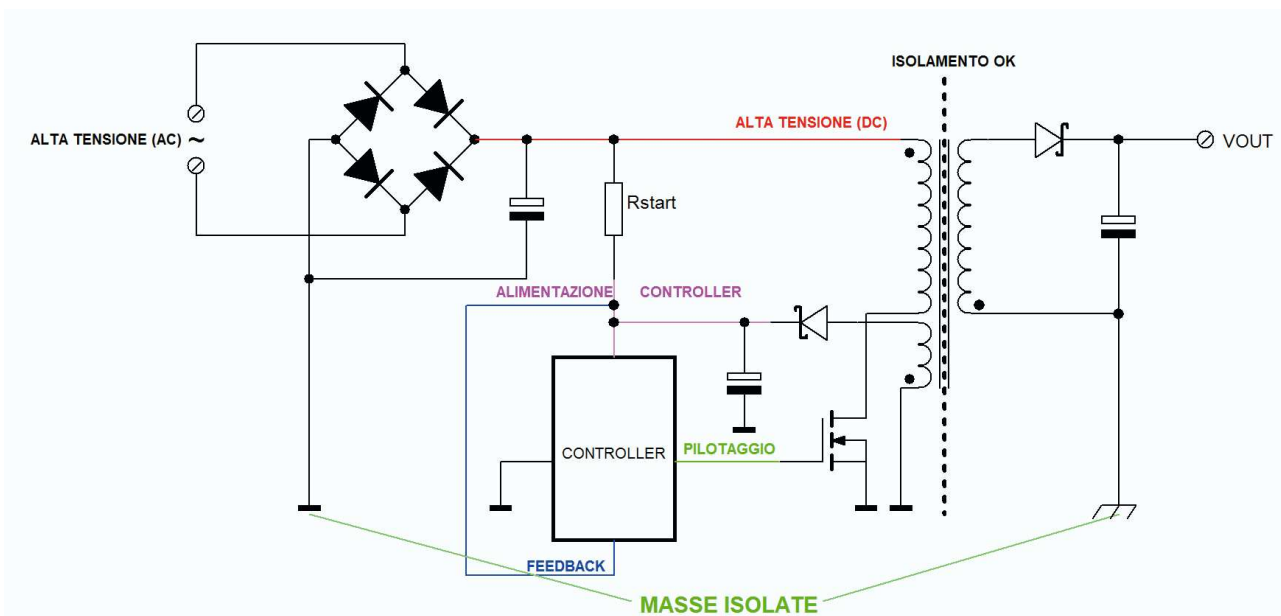



Figura 6 Schema di principio (II) di uno switching off-line ISOLATO

$$P_{OUT1,MAX} = V_{OUT1} * I_{OUT1,MAX} = 5 * 1 = 5W$$
$$P_{OUT2,MAX} = V_{OUT2} * I_{OUT2,MAX} = 12 * 0,2 = 2,4W$$
$$P_{OUT3,MAX} = |V_{OUT3}| * I_{OUT3,MAX} = 12 * 0,2 = 2,4W$$

La potenza totale sarà data semplicemente dalla somma delle precedenti:

$$P_{OUT,MAX} = P_{OUT1,MAX} + P_{OUT2,MAX} + P_{OUT3,MAX} =$$
$$= 5 + 2,4 + 2,4 = 9,8W$$

2. Ipotizziamo l'efficienza η pari al 75% e calcoliamo la potenza massima assorbita dall'ingresso:

$$P_{IN,MAX} = P_{OUT,MAX} / \eta = 9,8 / 0,75 = 13,067W$$

3. Ad ogni ciclo ON, l'energia immagazzinata nel trasformatore (cioè nell'induttanza del primario) si calcola con la:

$$E_L = 0,5 * Lp * (I_{p,MAX})^2$$

4. La **totalità** di questa energia **(dobbiamo imporre il funzionamento discontinuo)** viene trasferita al carico, per mezzo dei tre avvolgimenti secondari, ad ogni ciclo. Quindi la potenza trasferita, pari all'energia erogata in un secondo, è data da $E_L * f_{SW}$ (f_{SW} è la frequenza di switching) e deve essere uguale alla $P_{IN,MAX}$ calcolata al passo 2:

$$P_{IN,MAX} = E_L * f_{SW} = 0,5 * Lp * (I_{p,MAX})^2 * f_{SW}$$

5. Consideriamo ora che la $I_{p,MAX}$ dipende a sua volta (oltre che dall'induttanza del primario Lp), anche dalla tensione di ingresso e dal tempo di ON, secondo la nota formula:

$$I_{p,MAX} = (V_{IN} - V_{DROP}) / Lp * T_{ON}$$

(V_{DROP} è la caduta totale di tensione sul MOS e sulle resistenze).

6. Dai passi 4 e 5 precedenti, si ottiene con semplici manipolazioni algebriche:

$$P_{IN,MAX} = 0,5 * (V_{IN} - V_{DROP})^2 * T_{ON}^2 * f_{SW} / Lp$$

Nel caso peggiore, vale ovviamente la $V_{IN} = V_{IN,MIN}$. Ipotizziamo poi $V_{DROP} = 1V$ per le cadute resistive; come al solito, se alla fine questa ipotesi dovesse risultare troppo distante dalla realtà, sarà possibile tornare a questo passo sostituendo il valore più corretto. Nel caso di utilizzo di tipologia di controllo Current Mode, utilizziamo il valore massimo del 45% per il duty-cycle, quindi $T_{ON} = 0,45 * (1 / f_{SW})$, e sostituendo otteniamo:

$$P_{IN,MAX} = 0,5 * (18 - 1)^2 * 0,45^2 * (1 / f_{SW})^2 * f_{SW} / Lp$$

Questo valore deve essere maggiore o almeno uguale al valore desiderato, cioè 13,067W. Effettuando i calcoli e semplificando, si ottiene la:

$$Lp * f_{SW} \leq 2,239$$

Ormai sapete a memoria cosa fare: tabuliamo i risultati per varie frequenze (tabella 2).

f_{SW} [kHz]	Lp ≤ di [↔H]
20	112,0
30	74,6
40	49,8
50	44,8
60	37,3
70	32,0
80	28,0
90	24,9
100	22,4

Tabella 2 *Coppie frequenza/induttanza*

TIPOLOGIA "FLYBACK" MULTIUSCITA: SCELTA DEL NUCLEO

Per non costringervi ogni volta ad acquistare un nucleo/corpo bobina/mollette differenti, useremo quello della puntata precedente: il nucleo E20 (20x20x6mm). Ricordandovi dell'esistenza di varie versioni, a seconda del gap, otteniamo la tabella 3. Onde ridurre di dimensioni queste tediose tabelle, consideriamo solo le ferriti con gap di 0,25 e 0,5mm.

Ricordando che in uno switching il nucleo magnetico NON deve mai raggiungere la saturazione, cioè B deve sempre essere MINORE di B_{SAT}:

$$B_{MAX} = V_{IN} * T_{ON} / (N * Amin) \leq B_{SAT}$$

dove $V_{IN} = V_{IN,MIN} = 17V$, $T_{ON} = 0,45 * (1 / f_{SW})$, Amin = 31,6mm² per il nucleo E20. Supponiamo che B_{SAT} possa raggiungere al massimo 0,2 Tesla, cioè 200 mT. Riportiamo le densità di flusso calcolate, per ciascun nucleo/frequenza, in tabella 4.

Notate come il nucleo con gap = 0,25mm, entri in saturazione a qualunque frequenza!

Useremo quindi il nucleo con gap pari a 0,5mm, alla solita frequenza di 80kHz.
Il primario sarà ancora costituito da numero 16 spire, ma la corrente di picco aumenterà fino a 3,415A. Per questo motivo la *Rsense* deve scendere al valore di almeno 0,27Ω (nello switching singolo della puntata precedente era pari a 0,47Ω). Analogamente il valore del condensatore C1 è incrementato per sopportare il maggiore carico.
Con questa corrente di picco (che è sicuramente maggiore della RMS, ma lasciamo un certo margine di sicurezza), la tabella 1 forni-

f_{SW} [kHz]	Lp ≤ di [µH]	Nucleo B66311-G250-X127 A_L = 170nH SPIRE	Nucleo B66311-G500-X127 A_L = 106nH SPIRE
20	112,0	26	33
30	74,6	21	27
40	49,8	17	22
50	44,8	16	21
60	37,3	15	19
70	32,0	14	17
80	28,0	13	16
90	24,9	12	15
100	22,4	11	15

Tabella 3 *Numero di spire necessarie per ogni nucleo e frequenza*

f_{SW} [kHz]	Lp ≤ di [µH]	Nucleo B66311-G250-X127 A_L = 170nH B_{MAX} [mT]	Nucleo B66311-G500-X127 A_L = 106nH B_{MAX} [mT]
20	112,0	466	367
30	74,6	384	299
40	49,8	356	275
50	44,8	303	231
60	37,3	269	212
70	32,0	247	203
80	28,0	233	189
90	24,9	224	179
100	22,4	220	161

Tabella 4 *densità di flusso massima per ogni nucleo e frequenza*

sce un diametro filo di 1mm. Consiglio però di utilizzare due fili appaiati del diametro di 0,7mm, che hanno il vantaggio di essere più facilmente avvolgibili sul corpo bobina.

È interessante notare che, rispetto alla puntata precedente, all'aumento della potenza richiesta (a parità di nucleo) è corrisposto un aumento del gap, per far si che il nucleo magnetico riesca a sopportare l'incremento della densità di flusso, senza saturare.

Il calcolo del numero di spire dei secondari a 12V è immediato se si seguono questi semplici passi:

1. Il secondario regolato ha una tensione continua di 5V. Questo implica che la tensione di picco, ai capi del trasformatore e prima del diodo D2, sarà pari a 5 + 0,5 = 5,5V.
2. Se Ns1 = 5 spire corrispondono a 5,5V, quante spire saranno necessarie per ottenere 12 + 0,5 = 12,5?
   Usiamo il valore di 12,5V in modo da tenere conto della caduta sui diodi D1 e D3.
3. Il calcolo si traduce è una semplice proporzione: **5 : 5, 5 = Ns2 : 12,5** da cui si ricava **Ns2 = 11,4**. A questo punto potete decidere se avvolgere 11 spire ed ottenere una tensione leggermente minore di 12V, oppure se avvolgerne 12 ed ottenere una tensione leggermente superiore. Dallo schema è evidente quale è stata la mia scelta.
4. Poiché il terzo secondario è in serie al primo, basteranno Ns3 = 12 - 5 = 7 spire.

IL CIRCUITO

Anche questa volta ve l'ho fatto sudare...lo schema elettrico è praticamente IDENTICO a quello della puntata precedente, con la differenza che il trasformatore ha più secondari.

CONCLUSIONI

Abbiamo concluso con questa puntata la descrizione delle tipologie di pratico utilizzo per lo scopo di questo corso base. Ciò non vuole dire che abbiamo esaurito tutte le possibilità! Anzi, possiamo dire di avere solo cominciato, ma il tempo e lo spazio sono tiranni...Nelle ultime due puntate del corso descriverò in dettaglio un alimentatore *off-line* **commerciale**, scendendo nei dettagli della componentistica utilizzata, pezzo a pezzo. Vi chiedo quindi un ultimo sforzo, ne varrà la pena. Alla prossima!

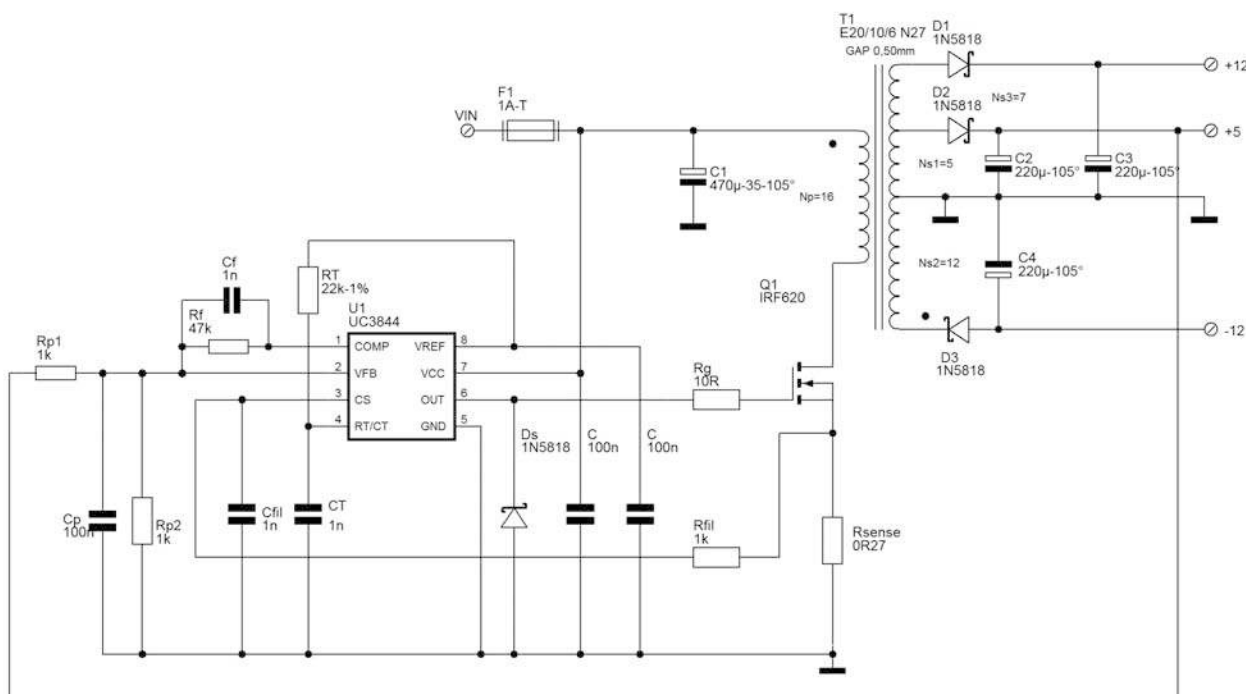



***Figura 7** Switching "flyback" 10W multiuscita, +5V, +12V, -12V*

Modulatore

Un circuito utile per "trasmettere" al vostro o ai vostri televisori un segnale audio/video proveniente da un lettore DVD, da una videocamera o da una qualsiasi sorgente video.

Tutti gli apparecchi video che troviamo in commercio oggigiorno sono muniti di almeno una presa SCART; alcuni, come i lettori DVD, ne hanno una, altri, come i videoregistratori o i decoder satellitari, ne hanno spesso due o addirittura tre, ma pochi di essi hanno al loro interno un modulatore RF capace di trasformare il segnale video e quello audio in un segnale "modulato" ricevibile da qualsiasi televisore, anche quelli sprovvisti di presa SCART.

Questa uscita RF è utile, come abbiamo già detto, quando abbiamo un televisore senza ingresso SCART, ma può tornarci comoda anche per "diffondere" il segnale sull'intero impianto antenna della nostra abitazione, in modo da vedere le immagini in diverse stanze anche distanti tra di loro.

L'utilizzo di questo circuito va però oltre le applicazioni descritte sopra: immaginate ad esempio di voler diffondere le immagini di una microcamera posta sopra la vostra porta di ingresso oppure in giardino in modo da tenere tutto sotto controllo da qualsiasi punto della vostra casa.

Applicando poi un mini-amplificatore ed una antenna all'uscita del circuito potrete trasformarlo in un vero e proprio "trasmettitore" capace di trasferire le immagini via etere senza bisogno di alcun cavo, potrete ad esempio tenere sotto controllo il vostro box auto anche se un po' distante dalla vostra abitazione senza stendere nessun cavo.

Attenzione però alle potenze perché vi sono delle leggi che regolamentano le emittenti e che impongono dei limiti per l'uso domestico privato, e poi non è bello far vedere agli altri i "fatti vostri"...

SCHEMA ELETTRICO

Il cuore del circuito che andiamo a presentare (figura 1) è un modulo costruito dalla SHARP al cui interno troviamo tutto il necessario per realizzare un completo modulatore audio-video in standard PAL.

Per far funzionare questo modulo abbiamo bisogno della sola alimentazione generale di 5 volt e di quella per il diodo varicap di 30 volt. Per quest'ultima abbiamo preferito utilizzare, invece di un alimentatore con uscita 30 volt, un piccolo elevatore che trasforma la tensione di base del circuito da 5 a 30 volt per permettere l'uso del modulatore anche in "portatile".

La tensione in ingresso (pari ad almeno 9 volt) attraversa prima, tramite il diodo D1, uno stabilizzatore ti tensione (IC2) che la

di Massimo Divito
div.massimo@tiscali.it

porta a 5 volt utile ad alimentare IC1 ed il modulo, e poi tramite l'impedenza-filtro L2 raggiunge IC3 che si occupa, come già detto, di elevare la tensione dai 5 ai 30, volt necessari al modulo per coprire l'intera gamma TV UHF.
All'accensione del circuito IC1 si preoccupa di leggere il canale impostato tramite il dip-switch SW1 e, dopo aver verificato che sia compreso tra il 21 ed il 69, invia al modulo, tramite il bus i2c, il comando per impostare la frequenza prescelta e conferma la corretta impostazione tramite l'accensione del diodo led DL1; i comandi I2C inviati dal PIC raggiungono il circuito integrato TDA8722 presente nel modulo che ha il compito di gene-

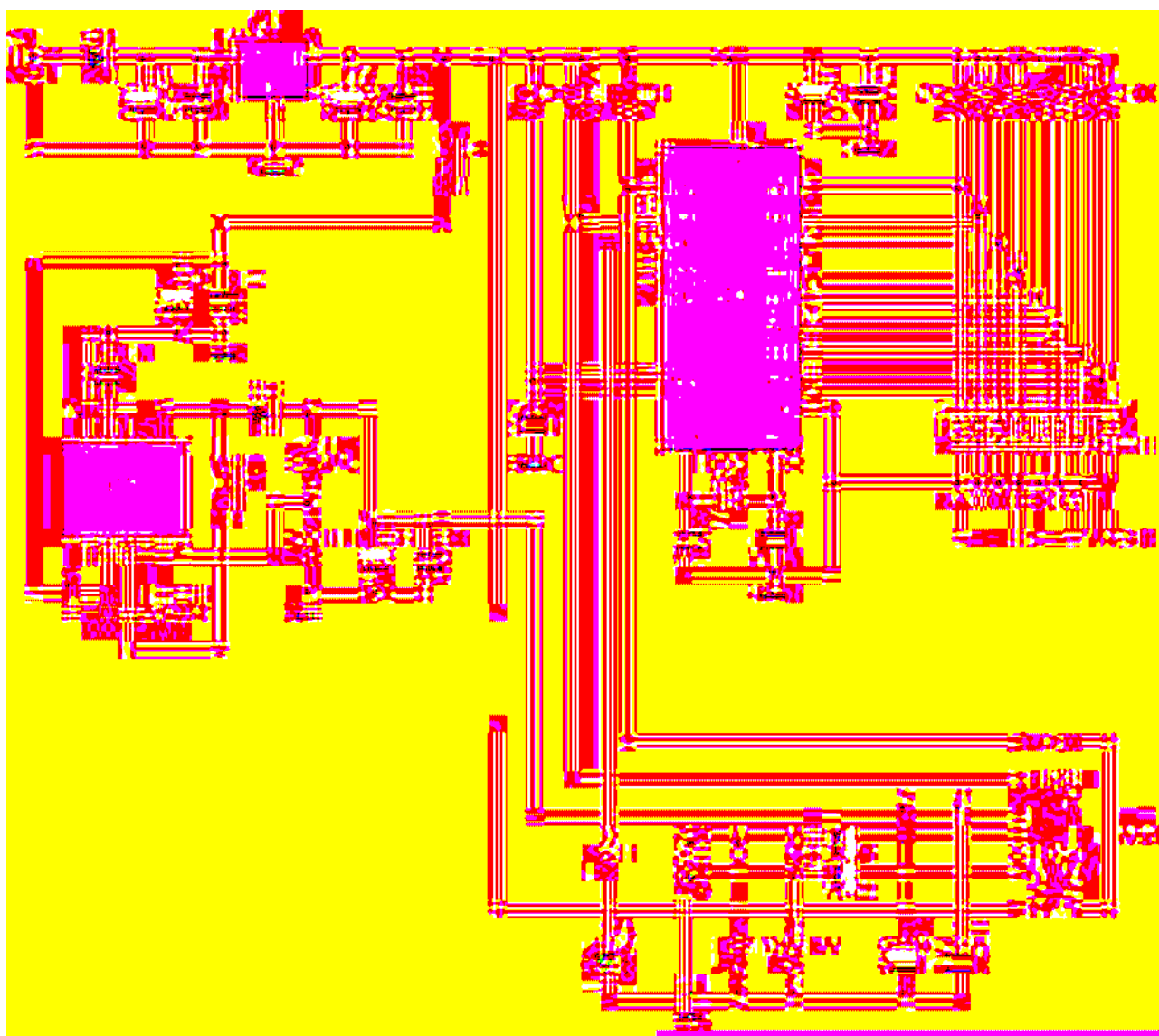



Figura 1 *Schema elettrico del modulatore*

rare la portante video, la sottoportante audio, ed a tenere stabile la frequenza impostata, visto che al suo interno contiene un completo circuito PLL.
Nel caso in cui il canale impostato non sia compreso tra il 21 ed il 69, il PIC (IC1) invia al modulo il comando di spegnimento, in modo da ridurre il rischio di interferenze RF ed avvisa la non corretta impostazione tramite lo spegnimento del diodo led DL1.
Tramite SW1 è anche possibile, chiudendo il contatto che fa capo alla porta RB7 del PIC, abilitare la funzione test del modulo che genera due barre verticali utili per una rapida sintonizzazione.

REALIZZAZIONE PRATICA

Per la realizzazione pratica abbiamo disegnato un apposito circuito stampato (figura 2) che potrete realizzare facilmente con il metodo della fotoincisione.
Bisognerà montare, seguendo il layout di figura 3, dapprima tutti i componenti più piccoli facendo molta attenzione a quelli polarizzati, si passerà poi al montaggio degli zoccoli per i circuiti integrati, utili per una rapida sostituzione e per un facile aggiornamento del firmware.
Fatto questo si procederà con il montaggio del modulo SHARP e del PIC, opportunamente programmato con il firmware che potete

Elenco componenti

Sigla	Valore	Sigla	Valore
R1, R2, R5÷R12	10 KΩ 1/4 W	C17	10 μF 25 V elettrolitico
R3	4,7 KΩ 1/4 W	C18	0,47 μF 16 V elettrolitico
R4, R15	220 Ω 1/4 W	IC1	PIC16F84
R13	47 KΩ 1/4 W	IC2	7805
R14	2 KΩ 1/4 W	IC3	MC34063A
R16	1,5 Ω 1/2 W	DL1	Diodo led 3 mm
R17	120 Ω	SW1	Dip-switch 8 posizioni
R18	22 KΩ 1/4 W	CN1	Connettore plug per cs
C1	1000 μF 25 V elettrolitico	CN2	Connettore RCA doppio per cs
C2, C3, C7, C10, C11, C13, C15	100 nF 63 V poliestere	XT	Quarzo 4 Mhz
C4, C8	100 μF 16 V elettrolitico	D1	1N4002
C5, C6	22 pF ceramico	D2	1N5819 o (1N4148)
C9	22 μF 16 V elettrolitico	L1	220 μH
C12	47 μF 16 V elettrolitico	L2	1 mH
C14	1 nF 63 V poliestere	DS1	Dissipatore in alluminio
C16	100 μF 16 V elettrolitico	MOD	Modulo UHF Sharp mod.E2251TA

CH	SW1							CH	SW1						
	2	3	4	5	6	7	8		2	3	4	5	6	7	8
21	OFF	ON	OFF	OFF	OFF	OFF	ON	46	OFF	OFF	ON	OFF	ON	ON	OFF
22	OFF	ON	OFF	OFF	OFF	ON	OFF	47	OFF	OFF	ON	OFF	ON	ON	ON
23	OFF	ON	OFF	OFF	OFF	ON	ON	48	OFF	OFF	ON	ON	OFF	OFF	OFF
24	OFF	ON	OFF	OFF	ON	OFF	OFF	49	OFF	OFF	ON	ON	OFF	OFF	ON
25	OFF	ON	OFF	OFF	ON	OFF	ON	50	ON	OFF	ON	OFF	OFF	OFF	OFF
26	OFF	ON	OFF	OFF	ON	ON	OFF	51	ON	OFF	ON	OFF	OFF	OFF	ON
27	OFF	ON	OFF	OFF	ON	ON	ON	52	ON	OFF	ON	OFF	OFF	ON	OFF
28	OFF	ON	OFF	ON	OFF	OFF	OFF	53	ON	OFF	ON	OFF	OFF	ON	ON
29	OFF	ON	OFF	ON	OFF	OFF	ON	54	ON	OFF	ON	OFF	ON	OFF	OFF
30	ON	ON	OFF	OFF	OFF	OFF	OFF	55	ON	OFF	ON	OFF	ON	OFF	ON
31	ON	ON	OFF	OFF	OFF	OFF	ON	56	ON	OFF	ON	OFF	ON	ON	OFF
32	ON	ON	OFF	OFF	OFF	ON	OFF	57	ON	OFF	ON	OFF	ON	ON	ON
33	ON	ON	OFF	OFF	OFF	ON	ON	58	ON	OFF	ON	ON	OFF	OFF	OFF
34	ON	ON	OFF	OFF	ON	OFF	OFF	59	ON	OFF	ON	ON	OFF	OFF	ON
35	ON	ON	OFF	OFF	ON	OFF	ON	60	OFF	ON	ON	OFF	OFF	OFF	OFF
36	ON	ON	OFF	OFF	ON	ON	OFF	61	OFF	ON	ON	OFF	OFF	OFF	ON
37	ON	ON	OFF	OFF	ON	ON	ON	62	OFF	ON	ON	OFF	OFF	ON	OFF
38	ON	ON	OFF	ON	OFF	OFF	OFF	63	OFF	ON	ON	OFF	OFF	ON	ON
39	ON	ON	OFF	ON	OFF	OFF	ON	64	OFF	ON	ON	OFF	ON	OFF	OFF
40	OFF	OFF	ON	OFF	OFF	OFF	OFF	65	OFF	ON	ON	OFF	ON	OFF	ON
41	OFF	OFF	ON	OFF	OFF	OFF	ON	66	OFF	ON	ON	OFF	ON	ON	OFF
42	OFF	OFF	ON	OFF	OFF	ON	OFF	67	OFF	ON	ON	OFF	ON	ON	ON
43	OFF	OFF	ON	OFF	OFF	ON	ON	68	OFF	ON	ON	ON	OFF	OFF	OFF
44	OFF	OFF	ON	OFF	ON	OFF	OFF	69	OFF	ON	ON	ON	OFF	OFF	ON
45	OFF	OFF	ON	OFF	ON	OFF	ON								

Tabella 1 *Selezione dei canali tramite l'impostazione di SW1*

scaricare dal sito di Fare Elettronica (www.farelettronica.com).
Per mezzo del dip-switch SW1 è possibile impostare la frequenza di trasmissione, riferendosi alla tabella 1; in alternativa si possono utilizzare due commutatori con codifica binaria, collegati secondo lo schema in figura 5, sui quali sarà possibile "leggere" direttamente il canale impostato.

Ricordate sempre che il modulo è di tipo "a doppia banda laterale", per cui, oltre alla portante a +5,5 MHz, ne presenta un'altra a –5,5MHz, è quindi consigliabile lasciare libero un canale al disotto della frequenza che si va ad utilizzare onde evitare fastidiose interferenze; in alternativa potete utilizzare un filtro trappola o un passa canale.

Figura 2 Circuito stampato in scala 1:1 (lato rame)

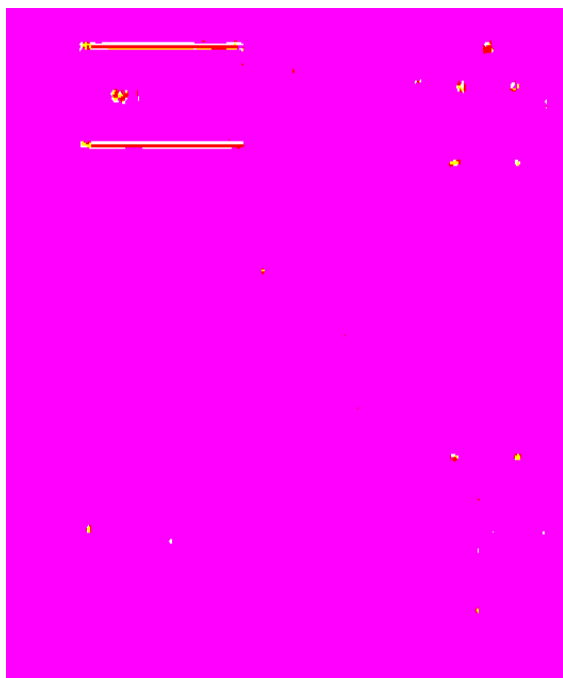



Figura 3 Piano di montaggio

CONCLUSIONI

Appena montato il circuito dovrebbe funzionare immediatamente. Collegate il segnale audio/video al connettore CN2, l'alimentazione al connettore CN1 (attenzione alla polarità) ed impostate il canale di trasmissione desiderato (SW1) seguendo la tabella 1.
Ponete il dip 1 di SW1 su ON e sintonizzate il vostro televisore fino a vedere perfettamente lo schermo di test.
A questo punto non vi resta che riportare il dip 1 in posizione OFF e godervi la vostra trasmissione.

Buon divertimento.

Figura 4 Schema per la connessione dei commutatori con codifica binaria

2005
CATALOGO
Generale

Vitamina C:

Cosa fare quando un programma che abbiamo scritto non funziona correttamente? Occorre cercare e correggere gli errori commessi. Questo procedimento può essere tutt'altro che semplice, e può richiedere uno sforzo notevole se non viene eseguito con i metodi e gli strumenti appropriati. In questa puntata ci occuperemo proprio di questo importante argomento.

INTRODUZIONE

Può sembrare strano, ma la fase di test e di debug del codice, soprattutto nel caso di progetti piuttosto complessi, può a volte richiedere più tempo di quello che è stato necessario per scrivere il codice. Questo è dovuto al fatto che i problemi che in genere si riscontrano quando si inizia a provare il programma possono essere molto vari, e soprattutto può essere difficile capire la loro origine e riuscire a risolverli. Per questi motivi la fase di debug del codice deve essere tenuta in grande considerazione, già dalle prime fasi di sviluppo del programma. Infatti, solo in questo modo è possibile limitare il suo impatto sui tempi di sviluppo (e quindi sui suoi costi). A tal fine è anche fondamentale conoscere ed utilizzare gli strumenti ed i metodi più appropriati per portare a termine questo compito, ed avere una discreta esperienza per individuare gli errori dai sintomi riscontrati. I prossimi paragrafi analizzano proprio questi aspetti.

INDIVIDUARE GLI ERRORI

Una volta completato un programma, normalmente lo si sottopone a una serie di test per controllare se è in grado di eseguire le operazioni per cui è stato creato e per verificare che esse siano realizzate correttamente (secondo le specifiche). Inoltre di solito ci si assicura anche che il programma sia relativamente tollerante a ingressi o condizioni di funzionamento non previste. Raramente questi test vengono superati subito, di solito si riscontrano dei problemi ed è necessario intervenire sul codice per risolverli.
I problemi che si possono riscontrare sono di due tipi: il programma può funzionare normalmente ma fornendo risultati errati, oppure il programma può bloccarsi in determinate condizioni o in maniera apparentemente casuale. Nel primo caso probabilmente si è commesso un errore nella scrittura dell'algoritmo o nella gestione dei dati.
Il secondo caso invece è il più insidioso, sia perché può dipendere da cause molto diverse, e quindi può richiedere maggiori sforzi per essere risolto, sia perché i malfunzionamenti possono non verificarsi subito, e restare quindi nascosti per molto tempo prima di potere essere osservati.

L'origine degli errori

In entrambi i casi la prima cosa che bisogna fare è cercare di individuare il problema, isolarlo (cioè capire quali fattori interni o esterni lo generano e quali invece sono ininfluenti), e tentare di riprodurlo, in modo da assicurarsi di averlo bene identificato. Riuscire ad osservare l'errore e le condizioni in cui esso si verifica in molti casi è il presupposto fondamentale per potere risolvere il problema. Molti dei metodi e degli strumenti presentati di seguito infatti, in ultima analisi, hanno proprio lo scopo di consentire un'osservazione più diretta e controllata dell'esecuzione del programma.

Sintomi tipici

Alcuni tipi di errori hanno delle manifestazioni

Tecniche di debug

di Antonio Di Stefano
a.distefano@farelettronica.com

abbastanza caratteristiche, è possibile quindi in molti casi individuare il tipo di problema anche soltanto dai suoi effetti.
Ad esempio alcuni tra gli errori più frequenti sono quelli che riguardano la gestione della memoria, in genere questi errori hanno conseguenze piuttosto disastrose, e portano ad un blocco del programma. Una prima categoria è generata da un uso non corretto di funzioni che allocano la memoria senza poi liberarla. Non bisogna pensare soltanto alla **malloc**, lo stesso effetto si può verificare anche con funzioni apparentemente più "inoffensive", quando queste sono richiamate un grande numero di volte (ad esempio quando si usano tecniche di ricorsione). In genere in questi casi la memoria si esaurisce, oppure vengono sovrascritte delle aree che contenevano dati, codice o addirittura lo stack, con conseguente blocco del programma. Ovviamente non c'è modo di accorgersi di questo problema fino a quando non si verifica un blocco apparentemente "casuale".
Un altro errore di memoria simile, è quello generato dall'esaurirsi dello spazio dedicato allo stack (stack overflow). Nei sistemi più semplici questa condizione non è evidenziata in nessun modo, l'unico effetto visibile è che i risultati improvvisamente diventano errati, il programma esegue operazioni non previste, ed infine può bloccarsi.
Anche l'uso scorretto dei puntatori può portare a conseguenze simili a quelle viste prima, anche se in genere più circoscritte. Se sono state fatte delle assegnazioni sbagliate, o si è commesso qualche errore di sintassi, è probabile che venga letta o scritta un'area di memoria sbagliata.
Questo può causare o degli errori persistenti nei dati, o il cattivo funzionamento del programma stesso (se viene sovrascritta una sua sezione). Quando invece il programma sembra funzionare correttamente, ma i risultati sono errati, chiaramente è stato commesso un errore nella scrittura dell'algoritmo. In molti casi, escludendo errori concettuali nell'implementazione dell'algoritmo, il problema risiede in un'assegnazione errata (variabile al posto di puntatore o viceversa, oppure uso di tipi di lunghezza errata), nel mancato uso di parentesi, o nella chiamata a funzioni in cui si sono utilizzati dei parametri errati o usati impropriamente.
Alcuni errori molto frequenti, ma abbastanza poco visibili sono causati dall'uso scorretto di variabili globali, o da una mancata inizializzazione esplicita delle variabili. In questo caso si possono verificare degli errori nei dati (dovuti ad una perdita di coerenza), comportamenti diversi dopo ogni reset, o il blocco del programma in alcuni loop (se alle variabili vengono assegnati valori non previsti si potrebbe non verificare la condizione di uscita).
Una volta individuato il tipo di errore, un buon metodo per isolare il problema è quello di provare ad escludere le sezioni di codice sospette (ad esempio commentandole, quando possibile).
Escludendo prima sezioni più grandi (intere funzioni), e poi più piccole (blocchi di codice e istruzioni), si può localizzare con precisione l'origine del problema.
Non sempre però è possibile capirne le ragioni e risolverlo, se non analizzando il flusso d'esecuzione nel suo complesso.

Prevenire gli errori

Sicuramente prevenire gli errori è molto più conveniente che doverli trovare e correggere in seguito. Questa affermazione può sembrare ovvia, ma non lo è affatto, in quanto richiede uno sforzo aggiuntivo in fase di scrittura del codice, che non sempre si è propensi a spendere. Tuttavia esso risulta indispensabile, soprattutto nel caso di progetti molto grandi, le cui parti sono sviluppate in parallelo da più programmatori. In questo caso può risultare molto costoso, se non impossibile, procedere per tentativi una volta terminato lo sviluppo dell'intero programma.
È quindi necessario assicurarsi che tutto il codice

prodotto funzioni bene da subito. Per fare questo è necessario in primo luogo che il programma sia sviluppato secondo un approccio *top-down*, cioè partendo da funzioni più piccole e semplici, che poi vengono unite per formare via via funzioni più complesse. Il vantaggio di questo approccio consiste nel fatto che è possibile testare in modo abbastanza completo le singole funzioni elementari, e quindi avere la certezza che funzioneranno quando poi verranno usate in un programma più complesso. Trovare un errore in una funzione elementare è molto più facile che trovarlo nel programma completo!
Ovviamente anche le interazioni tra le varie funzioni elementari devono essere testate poi, ma questo in genere risulta meno problematico.
Quando si scrive il codice si dovrebbe cercare di seguire sempre uno stile pulito e leggibile, in questo modo sarà più semplice trovare degli errori, perfino a chi non ha scritto il codice in prima persona. Inoltre la chiarezza del codice aiuta ad evitare molti errori dovuti proprio alla confusione del testo (parentesi mancanti o chiuse male, operatori con precedenza sbagliata…).
Anche l'uso di molti commenti facilita il debug, infatti gli errori possono essere individuati semplicemente verificando che il codice esegua le funzioni descritte in linguaggio naturale nei commenti stessi.
Per evitare problemi di ambiguità, o potenziali errori, è sempre meglio abbondare con le parentesi, sia nelle espressioni aritmetiche, sia nel passaggio di parametri "composti" a funzioni o macro. Non tutti i compilatori infatti considerano nello stesso modo le precedenze ed i raggruppamenti delle operazioni, quindi il codice dovrebbe sempre essere il meno ambiguo possibile per evitare problemi.
Una corretta indentazione invece aiuta a suddividere meglio il codice, e ad evitare di fare confusione nella chiusura dei blocchi, soprattutto quando ci sono molte sezioni lunghe e annidate. L'uso delle variabili globali dovrebbe essere limitato al minimo indispensabile, ed in ogni caso si dovrebbe prestare attenzione ed evidenziare le sezioni di codice in cui il contenuto di queste variabili viene modificato. Inoltre tutte le variabili dovrebbero essere inizializzate esplicitamente all'avvio del programma, questo per evitare che dopo un reset si possano verificare delle incoerenze o dei comportamenti imprevisti dovuti al fatto che in memoria è rimasto il valore precedente al reset.
Infine molta attenzione deve essere dedicata all'uso dei puntatori. È consigliabile verificare con cura le sezioni di codice che li utilizzano, ed assicurarsi anche di gestire i casi in cui ad essi possano essere assegnati valori non validi (molte funzioni di libreria ad esempio restituiscono un puntatore a NULL in caso di errore, e questo ovviamente non deve essere usato come puntatore valido).

METODI DI DEBUG

Quando un programma è stato completato e deve essere eseguito sulla macchina target si riscontrano subito alcune difficoltà: è difficile capire se il programma sta funzionando bene, e sapere in ogni momento cosa sta eseguendo e come. Infatti in molti casi non basta sapere solo se i risultati finali sono corretti, ma occorre anche seguire l'evoluzione del programma e le singole operazioni.
Questo problema è particolarmente sentito quando si programmano dei sistemi embedded, che interagiscono con segnali e periferiche molto velocemente, ed in maniera poco visibile dall'esterno.
Per ricavare maggiori informazioni si possono utilizzare appositi strumenti, che verranno descritti di seguito. Tuttavia esistono dei metodi piuttosto "primitivi", che si rivelano comunque molto utili nel caso in cui non si disponga di mezzi più sofisticati. Uno di questi, utile a capire se il programma sta eseguendo correttamente le operazioni più critiche, è quello di utilizzare una banale istruzione **printf** (o equivalente), per visualizzare su un display, o tramite comunicazione seriale delle informazioni quali ad esempio il valore di una variabile o la chiamata ad una funzione. Con questo sistema si può realizzare ad esempio un semplice *tracing* dell'esecuzione del programma: basta includere in ogni funzione un'istruzione *printf* che visualizzi il nome della funzione stessa, ed eventualmente alcuni suoi parametri, come mostrato nell'esempio seguente:

```
#include <stdio.h>

// - Prototipi -
int LeggiValore(void);
int Funzione1(void);
```

```
int Funzione2(int);

main() {
  int valore, passo;
  valore=0;
  passo=0;

  // - ciclo principale -
  switch(passo) {
    case(0):
      valore=LeggiValore();
      passo=1;
    break;
    case(1):
      passo=Funzione1();
    break;
    case(2):
      passo=Funzione2(valore);
    break;
    default:
      // *** DEBUG! ***
      printf(°∞Condizione imprevista!\n°±);
  }
}
int LeggiValore(void){
  ˇ
  // *** DEBUG! ***
  printf(°∞Funzione: LeggiValore\n°±);
  ˇ
}

int Funzione1(void){
  ˇ
  // *** DEBUG! ***
  printf(°∞Funzione: Funzione1\n°±);
  ˇ
}

int Funzione2(int){
  ˇ
  // *** DEBUG! ***
  printf(°∞Funzione: Funzione2\n°±);
  ˇ
}
```

Come si può vedere dal codice, il flusso d'esecuzione del programma è difficilmente prevedibile, soprattutto se il valore restituito dalle funzioni dipende da parametri legati all'hardware. L'uso delle *printf* permette di apprezzare in quale ordine vengono chiamate ed eseguite le funzioni. In altri casi può essere utile anche dichiarare alcune variabili globali ausiliarie per potere osservare da qualsiasi punto dei valori particolarmente significativi assunti da alcune variabili locali (nota: queste variabili globali non interferiscono con il funzionamento del programma, quindi non introducono "rischi" aggiuntivi).
Va notato che, se anche non si dispone di periferiche di output testuali, la stessa tecnica può essere applicata in modo ancora più essenziale utilizzando dei LED o dei segnalatori acustici presenti nel sistema, per rilevare il passaggio da un determinato punto del programma. Questa stessa tecnica si rivela particolarmente utile anche per comunicare all'esterno determinate condizioni in cui si trova il programma. Ad esempio lo stato di una linea di I/O può essere utilizzata per generare il segnale di trigger per un oscilloscopio o analizzatore logico, o per eseguire delle precise misure del tempo di esecuzione di alcune routine (si porta alto il livello della linea all'inizio della funzione, e lo si abbassa alla fine).
Un buon metodo per assicurarsi che il programma stia funzionando bene è quello di utilizzare delle "asserzioni". L'idea di base è la seguente: se il programma sta funzionando come voluto, in determinati punti devono essere verificate delle precise condizioni. Questa tecnica può essere parzialmente automatizzata usando appositi strumenti, ma può essere anche implementata manualmente. È sufficiente infatti porre nei punti voluti delle istruzioni che verificano le condizioni richieste, e forniscano un certo output, oppure interrompano l'esecuzione del programma se queste non sono verificate. In questo modo è relativamente facile stabilire il punto in cui si originano gli errori. Il codice seguente implementa la tecnica appena descritta:

```
int Divisione(int p, int q) {

   // *** DEBUG ***
   if (q==0) {
      printf(°∞Funzione Divisione: q=0!°±);
      while(1){}
   }
   return p/q;
}
```

L'istruzione *while(1)* è utilizzata per fermare l'esecuzione del programma. La sua utilità in fase di debug risiede nel fatto che a volte in presenza di errori il sistema potrebbe perdere il controllo del flusso di esecuzione e corrompere i dati presenti in memoria, o comunque sovrascriverli, impedendo di verificare le condizioni in cui si è verificato l'errore. Al contrario, potere osservare lo stato del sistema nel momento in cui si è verificato l'errore può fornire molte informazioni sulle sue cause.
È necessario a questo punto soffermarsi su un particolare. Il codice aggiunto durante la fase di debug può influenzare il programma, può ad esempio rallentarne l'esecuzione, ed in ogni caso comporta un incremento delle dimensioni del codice oggetto. È desiderabile pertanto eliminarlo una volta terminata la fase di debug. Per facilitare questo compito è necessario in primo luogo renderlo ben visibile, evidenziandolo almeno con dei commenti, come fatto sopra. Può essere utile anche impiegare dei "tag", cioè delle parole chiave facilmente riconoscibili, che possono essere ritrovate con una ricerca automatica nel testo. Una soluzione migliore è quella di utilizzare delle direttive del compilatore in modo da rendere possibile una compilazione condizionale, come mostrato nel codice seguente:

```
int Divisione(int p, int q) {

  // *** DEBUG ***
  #ifdef DEBUG
  if (q==0) {
    printf(°∞Funzione Divisione: q=0!°±);
    while(1){}
  }
  #endif
   return p/q;
}
```

In questo modo è possibile abilitare o disabilitare tutte le sezioni di codice appositamente inserite per il debug soltanto definendo o meno la macro DEBUG (cioè inserendo un #define DEBUG in uno dei moduli che ha visibilità massima).

STRUMENTI

Abbiamo visto fino ad ora alcuni metodi che possono essere utilizzati per eseguire il debug di un programma, considereremo ora alcuni strumenti che possono facilitare ed accelerare molto questo lavoro.

Source-level debugger e simulatori

Costituiscono il primo strumento da utilizzare dopo avere scritto il codice. Essi sono dei programmi che vengono eseguiti sull'host (spesso forniti assieme ai compilatori) e permettono di eseguire il codice scritto "simulando" il processore ed a volte anche alcune periferiche esterne ad esso collegate, o addirittura interi sistemi. In genere è possibile eseguire il codice per intero o in modalità passo-passo, ed avere la completa visibilità dei registri del processore, di quelli di eventuali periferiche, e della memoria di sistema (anch'essa simulata). Alcune delle funzioni più potenti che si possono utilizzare sono quelle legate ai *watch* ed ai *breakpoint*. I primi danno la possibilità di controllare costantemente il valore assunto da alcune variabili, i secondi permettono invece di bloccare il programma quando si verificano determinate condizioni, anche molto complesse. Utilizzando congiuntamente questi due strumenti è possibile seguire l'esecuzione del programma, e rendersi conto subito di eventuali errori, della loro posizione, e delle loro cause.
Ad esempio si può impostare un breakpoint su una condizione collegata ad uno degli errori riscontrati, e simulare il programma.
Quando la condizione sarà verificata, la simulazione si bloccherà, e sarà possibile osservare il punto del programma che ha originato la condizione di errore, controllare il valore delle variabili e dei registri, o eseguire passo-passo il programma. In questo modo in genere è possibile individuare con precisione le cause degli errori.
L'uso dei simulatori può quindi accelerare molto la fase iniziale di test e debug, permettendo di correggere subito molti errori. L'accuratezza dei simulatori può variare molto, alcuni simulano soltanto l'esecuzione delle istruzioni, altri eseguono delle simulazioni dell'hardware accurate al singolo ciclo di clock, e quindi permettono di scoprire anche problemi che si manifestano a livello più basso.

Hardware debugger

Anche quando il codice è stato verificato tramite un simulatore, può succedere che si verifichino degli errori legati alle tempistiche reali del sistema, ed all'interazione con l'hardware e soprattutto con i segnali esterni. In questi casi un aiuto può essere dato dai debugger hardware. Essi sono degli strumenti (che devono essere supportati dal sistema) che consistono essenzialmente in tre elementi: un'interfaccia di comunicazione tra il target e l'host (RS232, JTAG…), un piccolo programma (chiamato "monitor") che viene aggiunto al codice prodotto e caricato sul sistema target, ed un programma "front-end" eseguito sul computer host, che permette di inviare comandi al sistema target e di visualizzarne i dati, comunicando con il programma monitor. Alcuni sistemi non necessitano di un programma monitor poiché sono dotati di un dispositivo hardware apposito, che svolge le stesse funzioni (tra l'altro in maniera più efficiente, e meno invasiva).
Il debugger hardware permette di interrompere l'esecuzione del programma sul target, e compiere operazioni simili a quelle descritte prima a proposito dei simulatori, cioè leggere e scrivere il valore dei registri o della memoria, di eseguire passo-passo il programma, e di impostare dei semplici breakpoint.
In genere l'interazione avviene con comandi impartiti manualmente del tipo "leggi il registro x", "leggi n byte a partire dalla locazione di memoria y", "scrivi b nella locazione z", etc. Dopo avere esaminato i valori di interesse, ed averli eventualmente modificati, è possibile anche riprendere l'esecuzione del programma.
I degugger sono uno strumento molto utile e potente per controllare l'esecuzione del programma sul sistema reale, e per testare il corretto funzionamento delle periferiche del sistema (che si possono accedere agendo sullo spazio di memoria o di I/O del processore), e la loro interazione con il programma. Per aumentare le possibilità di intervento in molti casi può essere utile impiegare assieme al debugger alcuni dei metodi software visti prima. In particolare bloccare il programma quando si verificano determinate condizioni è utile per potere leggere dei valori in quella particolare situazione.
Uno dei debugger più noti è lo GNU Debugger (GDB), che è disponibile gratuitamente per moltis-

simi processori e sistemi, e per questo è diventato in un certo senso uno "standard". Molti altri debugger imitano lo stile ed il funzionamento del GDB.

In-Circuit Debuggers/Emulators

Sono degli strumenti estremamente potenti, e fino a qualche tempo fa molto costosi, che ultimamente si stanno diffondendo molto, fino ad essere integrati anche in piccoli microcontrollori. Essi in pratica mettono insieme la funzionalità dei simulatori e dei debugger hardware, ossia permettono di avere una completa visibilità e controllabilità dell'esecuzione del codice sul sistema reale. Utilizzando l'In-Circuit Debugger (abbreviati spesso ICD o ICE) è possibile seguire in tempo reale (o quasi) l'evoluzione del programma, monitorare costantemente il valore dei registri, il contenuto della memoria, la presenza di interruzioni, e soprattutto è possibile impostare complessi breakpoint hardware, simili a quelli utilizzabili con i simulatori. Queste funzioni facilitano molto l'individuazioni di errori o malfunzionamenti dell'intero sistema, e permettono di capirne facilmente le cause. Ad esempio, se si nota che una funzione di allarme non viene richiamata quando dovrebbe, si potrebbe scoprire che i valori forniti dal convertitore A/D non superano mai la soglia impostata, a causa magari di un guadagno troppo basso nella parte analogica. Oppure se si vuole capire perché a volte alcuni dati assumono valori errati, si può impostare un breakpoint con la condizione "ferma in caso di scrittura alla locazione x", e quindi si possono individuare facilmente le istruzioni che originano l'errore.

Uso di oscilloscopi ed analizzatori logici

Può sembrare strano, ma spesso per capire i motivi per cui un programma non funziona come dovrebbe, non è sufficiente eseguire il debug sul solo software, ma è necessario estenderlo anche all'hardware. In particolare in molti casi l'evoluzione del programma è fortemente influenzata dall'interazione con segnali esterni. A volte l'origine di alcuni malfunzionamenti è dovuta al fatto che alcuni segnali non hanno le caratteristiche previste, anche solo per brevi periodi di tempo. Per eseguire queste verifiche è necessario monitorare congiuntamente i segnali hardware e l'esecuzione del software.
Per fare questo si utilizzano prevalentemente due strumenti: gli oscilloscopi e gli analizzatori logici.
I primi sono più indicati per seguire l'andamento di segnali analogici, ed in particolare rilevarne le caratteristiche in corrispondenza dell'esecuzione di alcune routine. Per fare questo è possibile inviare ad un canale dell'oscilloscopio il segnale analogico in questione, e ad un altro il livello di un piedino di I/O pilotato in modo da fornire un riferimento temporale. Questa tecnica può essere utilizzata anche per verificare che i segnali generati dal programma rispettano le temporizzazioni previste. Quando invece occorre verificare l'andamento e lo stato di diversi segnali digitali, lo strumento più indicato è l'analizzatore logico. Esso può registrare i segnali presenti su diverse linee (anche 32 o 64) durante un certo intervallo di tempo. La registrazione viene eseguita ciclicamente su una memoria, e si interrompe quando si verifica una precisa condizione (evento di trigger), che può essere data da una certa combinazione presente sulle linee, o dallo stato di alcune di esse. In questo modo è possibile non solo visualizzare le caratteristiche dei segnali precedenti l'evento di trigger, ma anche le loro temporizzazioni con estrema precisione. Anche in questo caso un I/O libero può essere utilizzato per sincronizzare gli eventi o per fornire un riferimento. Utilizzando l'analizzatore logico si possono facilmente scoprire fenomeni come glitch, contentions e termporizzazioni errate. È anche possibile seguire le comunicazioni su un bus di sistema o su una linea I^{2}C per verificare la loro correttezza, o scoprire che alcuni dispositivi non rispettano le temporizzazioni dichiarate (ad esempio generano segnali di clock a frequenza variabile o con duty-cycle diverso dal 50%).

CONCLUSIONE

Molti dei suggerimenti forniti possono essere applicati con buoni risultati al debug dei propri programmi, ma per sfruttarne al massimo le potenzialità occorre un po' di pratica.
Non esistono infatti delle regole e dei metodi precisi per eseguire il debug, è quindi necessaria una buona dose di perspicacia ed esperienza per individuare e correggere gli errori in breve tempo.
Nella prossima puntata verrà preso in considerazione un argomento che in un certo senso è spesso causa di errori e malfunzionamenti se non ben curato: la gestione delle interruzioni in C.

Rispondi correttamente al quiz e potrai vincere un abbonamento omaggio a

Fare elettronica

Partecipare è semplicissimo:

rispondi al quesito seguendo il regolamento e, se la risposta si rivelerà esatta, potrai vincere un abbonamento omaggio (o il rinnovo qualora fossi già abbonato) a Fare Elettronica. Ogni mese sulle pagine della rivista troverai la soluzione del quesito del mese precedente e il nome del vincitore di uno dei quesiti pubblicati. Per tutti i partecipanti è previsto comunque un coupon del 10% di sconto utilizzabile per un acquisto nello shop di www.farelettronica.com.

Quesito - LST24408

Si consideri un chip DRAM da 4Mbx4:

1) I pin di indirizzo sono:
- A. 11
- B. 12
- C. 22
- D. 24

2) Le celle elementari di memorizzazione sono:
- A. 1048576
- B. 4194304
- C. 16777216
- D. 67108864

3) Quanti transistori contiene:
- A. circa 4 milioni
- B. circa 16 milioni
- C. circa 18 milioni
- D. circa 64 milioni

Scadenza: il termine ultimo per rispondere è il 30 Ottobre 2005

Regolamento

1) Il quiz è aperto a tutti i lettori.

2) Saranno considerate esclusivamente le risposte pervenute entro la scadenza indicata nel quesito.

3) Inviare la risposta compilando il modulo su **www.farelettronica.com/lesotutte**.

Riflettori su... MikroC:

Un nuovissimo compilatore C per PICmicro® completo di IDE ed una moltitudine di esempi di programmazione. È questa la nuova proposta di Mikroelektronika, la casa produttrice dell'ormai famoso MikroBasic e della rivoluzionaria easyPIC2. Ecco tutti gli strumenti che mikroC mette a disposizione dello sviluppatore.

GLI STRUMENTI

La versione corrente, al momento della presente pubblicazione, è la v.2.1 che gira sotto Windows® 98/2000/NT/XP. Il compilatore supporta il linguaggio assembler ed ANSI C ed è in grado di produrre file Assembler, binari o conformi allo standard Intel HEX.

Code Explorer

Code Explorer consente il monitoraggio di variabili, funzioni, procedure e molto altro. Un doppio click sul nome della variabile o della funzione nella finestra *code explorer* e si è subito rimandati alla relativa riga del codice sorgente.

Code Editor

MikroC è dotato di un editor intuitivo adatto anche a chi è alle prime esperienze con la programmazione. *Code Assistant* permette, digitando le prime tre lettere di una parola, di avere suggerimenti su tutte le possibilità compatibili con quanto già digitato. Il *Parameter Assistant* permette invece di visualizzare come suggerimento i parametri attesi da una certa funzione o routine. È possibile abilitare anche la correzione automatica degli errori aggiungendo anche parole personalizzate oltre ai comandi specifici del linguaggio C.

Il *Code Editor* è configurabile quindi è possibile associare diversi colori ai comandi, alle variabili, alle costanti, ecc.

Figura 1 *Code Explorer*

Debugger

MikroC include un eccellente debugger che permette l'individuazione di errori in modo veloce. Una volta compilata l'applicazione è possibile lanciare il debugger che consente l'esecuzione passo-passo del programma, l'introduzione di breakpoints e può interagire con la watch window (figura 2) che consente di monitorare lo stato delle voci di programma durante la simulazione. Vengono visualizzati tutti i registri del PICmicro® e le variabili definite dall'utente, mostrando il relativo indirizzo ed il valore contenuto all'istante corrente.

Statistiche

Le funzioni di statistica sono estremamente utili per il monitoraggio dell'occupazione della memoria del micro. MikroC consente 6 tipologie di statistiche differenti:

Memory Usage Window – Fornisce una panoramica, sotto forma di istogramma, della RAM e ROM utilizzata.

Procedures (Graph) Window – un istogramma che mostra l'allocazione di memoria da parte delle funzioni e delle procedure.

Procedure (Locations) Window – Mostra la collocazione delle procedure e delle funzioni nella memoria del micro.

Procedure (Details) Window – Mostra un albero di dettagli per ciascuna procedura o funzione. Vengono mostrati la dimensione, l'indirizzo di

Un nuovo compilatore C per PICmicro®

inizio e di fine, la frequenza di chiamata nel programma, il tipo di dato del risultato, ecc.
RAM Window – Un riepilogo di tutti i registri GPR e SFR con i loro indirizzi e contenuti, oltre a tutte le altre variabili definite dall'utente.
ROM Window – La lista di tutti gli opcodes ed i relativi indirizzi presentati nella forma più comprensibile per l'utente.

ROUTINES DI LIBRERIA

Esistono numerose routines nelle librerie di MikroC pronte da utilizzare. Al momento le librerie disponibili sono le seguenti:

- ADC Library
- CAN Library
- CANSPI Library
- Compact Flash Library
- EEPROM Library
- Ethernet Library
- Flash Memory Library
- Graphic LCD Library
- I²C Library
- Keypad Library
- LCD Library
- LCD8 Library
- Manchester Code Library
- Multi Media Card Library
- OneWire Library
- PS/2 Library
- PWM Library
- RS-485 Library
- Secure Digital Library
- Software I_C Library
- Software SPI Library
- Software UART Library
- Sound Library
- SPI Library
- USART Library
- USB HID Library
- Util Library
- Conversions Library
- Trigonometry Library

Oltre alle librerie, vengono forniti numerosi esempi di programmazione che illustrano l'uso di tutte le funzioni di libreria.

RIFERIMENTI

Per ulteriori informazioni sul compilatore *MikroC*, consultare il sito www.elettroshop.com da cui è possibile scaricare la manualistica e la versione demo del compilatore.

Elettroshop
Via Cadorna, 27/31 – 20032 Cormano (MI)
Tel. 02.66504794 Fax 02.66508225
info@elettroshop.com – www.elettroshop.com

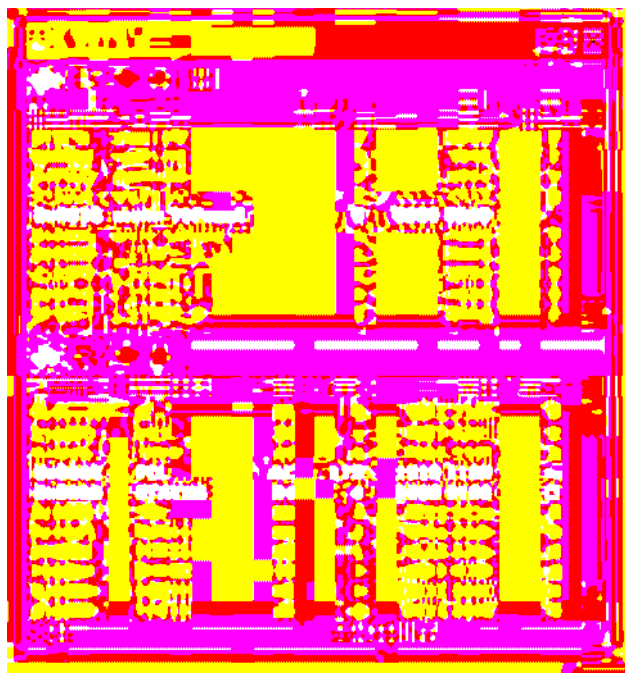



Figura 2 Watch window

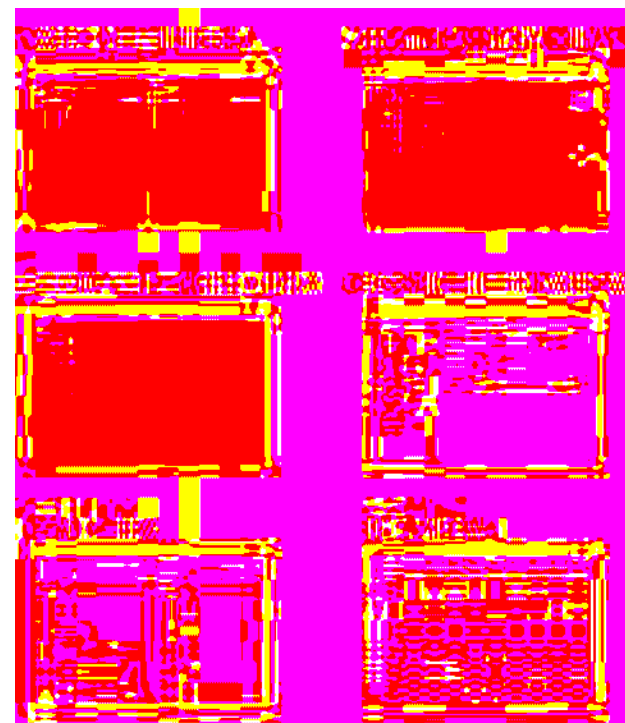



Figura 3 Statistiche di MikroC

Si approfitto della fantastica occasione e mi abbono a 11 numeri di Fare Elettronica e pagherò solo € 45,00

DATI PERSONALI

Nome ....

Cognome ....

Via .... n° ....

Cap .... Città .... Prov ....

Tel .... Fax ....

Email ....

Ragione Sociale ....

P.Iva .... ❍ Ricevuta

Privacy. Ai sensi del Decr. Lgs. 196/2003 la informiamo che i dati trasmessi verranno impiegati coi principali scopi di indagini di mercato e nelle modalità previste dallo stesso, prevalentemente con mezzi informatici. Il conferimento, di norma facoltativo, è obbligatorio per permettere il rapporto commerciale. È in ogni caso fatto diritto dell'interessato esercitare i propri diritti, nei modi previsti dal "Titolo II art. 7" della legge sopra citata, scrivendo a Inware Edizioni Via Cadorna 27 – 20032 Cormano o trmite email a info@inwaredizioni.it

ABBONATI SUBITO

Compila, ritaglia e spedisci via fax questo coupon allo **02-66508225**

Spedisci questa pagina in busta chiusa a:
INWARE Edizioni
Via Cadorna, 27/31 - 20032 Cormano (MI)

Chiamaci allo **02-66504794**

Abbonati on-line sul sito
www.farelettronica.com/abbonamento

MODALITÀ DI PAGAMENTO

❍ **CARTA DI CREDITO**

❑ American Express ❑ Visa ❑ Mastercard

Titolare ....

n° .... scad ....

❍ **VERSAMENTO SUL CCP 22790232**

Allegare la ricevuta (o copia) del versamento intestato a Inware Srl, indicando nella causale: "Abbonamento Fare Elettronica"

❍ **BONIFICO BANCARIO**

Appoggiarlo su: **Poste Italiane - CIN: Z - ABI: 07601 CAB: 01600 - C/C: 000022790232** intestato ad **Inware srl**

❍ **ALLEGO UN ASSEGNO**

intestato a Inware Srl

Firma ....

Abbonati subito!

- ✔ **Risparmierai** ben 15,50 euro sul prezzo di copertina
- ✔ Avrai la garanzia del **prezzo bloccato** per un anno
- ✔ La rivista ti sarà **recapitata comodamente** a casa
- ✔ Compreso con l'abbonamento (o il rinnovo) riceverai un **buono sconto del 20%** per un tuo prossimo acquisto sul sito www.farelettronica.com e, insieme alla merce ordinata, ti sarà recapitato l'esclusivo portapenne di Fare Elettronica

INWARE EDIZIONI